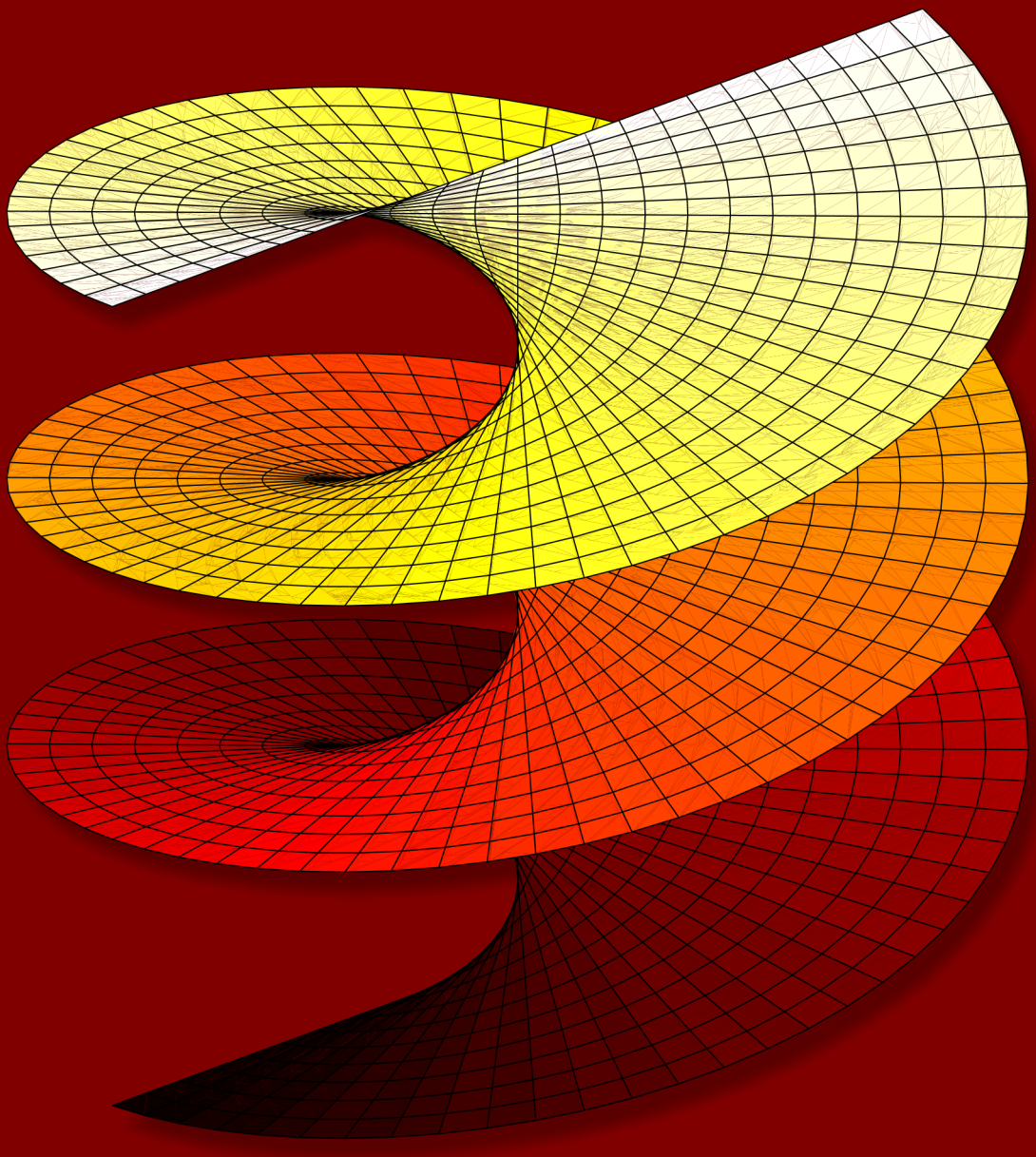
Introduction to GNU Octave

A brief tutorial for linear algebra and calculus students



Jason Lachniet

Introduction to GNU Octave

A brief tutorial for linear algebra and calculus students

Jason Lachniet
Wytheville Community College

Third Edition

© 2020 by Jason Lachniet (CC-BY-SA)

This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License. To view a copy of this license, visit http://creativecommons.org/licenses/by-sa/4.0/ or send a letter to Creative Commons, PO Box 1866, Mountain View, CA 94042, USA.

Edition 3

Download for free at:
https://www.wcc.vccs.edu/sites/default/files/Introduction-to-GNU-Octave.pdf.

Contents

Contents **vii**

Preface **ix**

1 Getting started **1**
1.1 Introduction . . . 1
1.2 Navigating the GUI . . . 3
1.3 Matrices and vectors . . . 4
1.4 Plotting . . . 9
Chapter 1 Exercises . . . 14

2 Matrices and linear systems **17**
2.1 Linear systems . . . 17
2.2 Polynomial curve fitting . . . 23
2.3 Matrix transformations . . . 27
Chapter 2 Exercises . . . 33

3 Single variable calculus **39**
3.1 Limits, sequences, and series . . . 39
3.2 Numerical integration . . . 43
3.3 Parametric, polar, and implicit functions . . . 46
3.4 The symbolic package . . . 50
Chapter 3 Exercises . . . 57

4 Miscellaneous topics **59**
4.1 Complex variables . . . 59
4.2 Special functions . . . 61
4.3 Statistics . . . 63
Chapter 4 Exercises . . . 70

5 Eigenvalue problems **73**
5.1 Eigenvectors . . . 73
5.2 Markov chains . . . 74
5.3 Diagonalization . . . 78
5.4 Singular value decomposition . . . 82
5.5 Gram-Schmidt and the QR algorithm . . . 88
Chapter 5 Exercises . . . 93

6 Multivariable calculus and differential equations **97**
6.1 Space curves . . . 97
6.2 Surfaces . . . 99
6.3 Multiple integrals . . . 108
6.4 Vector fields . . . 114
6.5 Differential equations . . . 117
Chapter 6 Exercises . . . 124

7 Applied projects **127**
7.1 Digital image compression 127
7.2 The Gini index 130
7.3 Designing a helical strake 133
7.4 3D-printing 136
7.5 Modeling a cave passage 143
7.6 Modeling the spread of an infectious disease 152

A MATLAB compatibility **157**

B Octave command glossary **159**

References **163**

Index **165**

Preface

This short book is not intended to be a comprehensive manual (for that refer to [3], which is indispensable). Instead, what follows is a tutorial that puts Octave to work solving a selection of applied problems in linear algebra and calculus. The goal is to learn enough of the basics to begin solving problems with minimum frustration. Note that minimum frustration does not mean *no* frustration. Be patient!

Above all, our objective is simply to enhance our understanding of calculus and linear algebra by using Octave as a tool for computations. As we work through the mathematical concepts, we will learn the basics of programming in Octave. Note that while we deal with plenty of useful numerical algorithms, we do not address issues of accuracy and round-off error in machine arithmetic. For more details about numerical issues, refer to [1].

How to use this book

To get the most out of this book, you should read it alongside an open Octave window where you can follow along with the computations (you will want paper and pencil, too, as well as your math books). To get started, read Chapter 1, without worrying too much about any of the mathematics you don't yet understand. After grasping the basics, you should be able to move into any of the later chapters or sections that interest you.

Every chapter concludes with a set of problems, some of which are routine practice, and some of which are more involved. Chapter 7 contains a series of applied projects. Most examples assume the reader is familiar with the mathematics involved. In a few cases, more detailed explanation of relevant theorems is given by way of motivation, but there are no proofs. Refer to the linear algebra and calculus books listed in the references for background on the underlying mathematics. In the spirit of openness, all references listed are available for free under GNU or Creative Commons licenses and can be accessed using the links provided.

MATLAB

The majority of the code shown in this book will work in MATLAB as well as Octave. This guide can therefore also be used an introduction to that software package. Refer to Appendix A for some notes on MATLAB compatibility.

Formatting

Blocks of Octave commands are indented and displayed with special formatting as follows.

```
>> % example Octave commands:
>> x = [-3 : 0.1 : 3];
>> plot(x, x.^2);
>> title('Example plot')
```

The same formatting is used for commands that appear inline in the text. Comments used to explain the code are preceded by a %-sign and shown in green. Function names are highlighted in magenta. Strings (text variables) are highlighted in purple. The Octave prompt is shown as ">>". This color coding visible in the PDF e-book is not essential to understand the text. Thus the print version is in black and white, which keeps the price reasonable.

Octave scripts and function files (.m-files) are shown between horizontal rules and are labeled with a title, as in the following example. These are short programs in the Octave language.

Octave Script 1: Example

```
1 % This is an example Octave script (.m-file)
2 t = linspace(0, 2*pi, 50);
3 x = cos(t);
4 y = sin(t);
5
6 % plot the graph of a unit circle
7 plot(x, y);
```

Note that line numbers are for reference purposes only and are not part of the code

If you are reading the electronic PDF version, there are numerous hyperlinks throughout the text that link back to other parts of the text, or to external urls. There is a set of bookmarks to each chapter and section that can be used to easily navigate from section to section. Open the bookmark link in your PDF viewer to use this feature.

Theorems and example problems are numbered sequentially by chapter and section (e.g., Theorem 5.3.4 is the fourth numbered item in Chapter 5, Section 3).

> Solutions to the many example problems are offset with a bar along the left side of the page, as shown here. A box signifies the end of the example. □

Feedback

If you use the book and find it helpful, please consider leaving a review on the Amazon.com product page or at the UMN Open Textbook Library listing. Send corrections and suggestions to the author at jlachniet@wcc.vccs.edu.

Revision history

2017	First edition • Written for Octave version 4.0.
2019	Second edition • Updated to reflect changes implemented in Octave through version 4.4, including the addition of a variable editor, migration of some statistical functions to the statistics package, and the addition of Matlab-compatible ODE solvers to the Octave core. • New material added on implicit plots, complex variables, matrix transformations, and symbolic operations. • Addition of several new exercises and a new chapter containing a set of applied projects suitable for linear algebra and calculus students.
2020	Third edition • Updated for Octave version 5.2. • More extensive three-dimensional plotting examples. • Increased coverage of the symbolic package. • Numerous small edits throughout to improve clarity of exposition, based on reviewer and student feedback. • Multiple new or revised exercises throughout, and one new project. • A (slightly expanded) treatment of statistics has moved to a new Chapter 4, along with a few other miscellaneous topics, leaving only core single variable calculus topics in Chapter 3. Subsequent chapters are renumbered to accommodate this update. What is now Chapter 6 contains only multivariable calculus and differential equations.

Chapter 1

Getting started

1.1 Introduction

1.1.1 What is GNU Octave?

GNU Octave is free software designed for scientific computing. It is intended primarily for solving numerical problems. In linear algebra, we will use Octave's capabilities to solve systems of linear equations and to work with matrices and vectors. Octave can also generate sophisticated plots. For example, we will use it in vector calculus to plot vector fields, space curves, and three dimensional surfaces. Octave is mostly compatible with the popular "industry standard" commercial software package MATLAB, so the skills you learn here can be applied to MATLAB programming as well. In fact, while this guide is meant to be an introduction to Octave, it can serve equally well as a basic introduction to MATLAB.

What is "GNU?" A *gnu* is a type of antelope, but *GNU* is a free, Unix-like computer operating system. GNU is a recursive acronym that stands for "GNU's not Unix." GNU Octave (and many other free programs) are licensed under the GNU General Public License: http://www.gnu.org/licenses/gpl.html.

From www.gnu.org/software/octave:

> GNU Octave is a high-level interpreted language, primarily intended for numerical computations. It provides capabilities for the numerical solution of linear and non-linear problems, and for performing other numerical experiments. It also provides extensive graphics capabilities for data visualization and manipulation. Octave is normally used through its interactive command line interface, but it can also be used to write non-interactive programs. The Octave language is quite similar to MATLAB so that most programs are easily portable.

Octave is a fully functioning programming language, but it is not a general purpose programming language (like C++, Java, or Python). Octave is numeric, not symbolic; it is not a computer

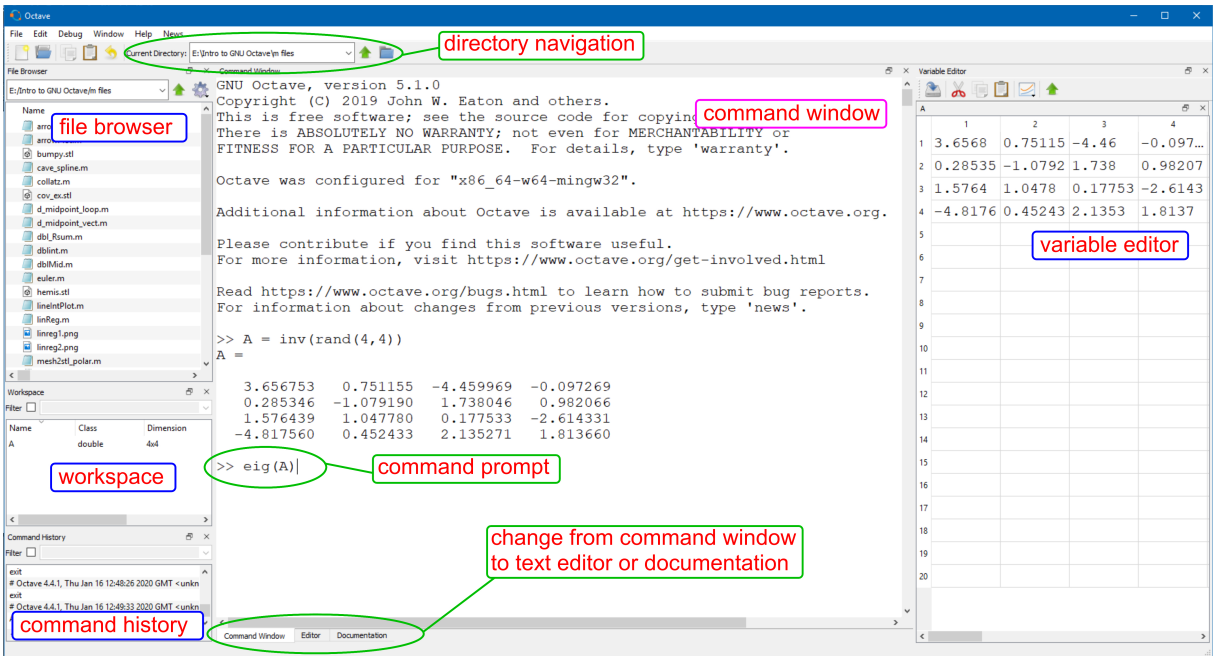


Figure 1.1: Windows Octave GUI

algebra system (like Maple, Mathematica, or Sage)[1]. However, Octave is ideally suited to all types of numeric calculations and simulations. Matrices are the basic variable type and the software is optimized for vectorized operations.

1.1.2 Installing Octave

It's free! Octave will work with Windows, Macs, or Linux. Go to https://www.gnu.org/software/octave/download.html and look for the download that matches your system. For example, Windows users can find an installer for the current Windows version at https://ftp.gnu.org/gnu/octave/windows/. Manual installation can be tricky, so look for the most recent `.exe` installer file and run that. Installation in most Linux systems is easy. For example, in Debian/Ubuntu, run the command `sudo apt-get install octave`. If you find Octave useful, consider making a donation to support the project at https://www.gnu.org/software/octave/donate.html.

Beginning with version 4.0, Octave uses a graphical user interface (GUI) by default. When you start Octave, you should see something like Figure 1.1.

The user can customize the arrangement of windows. By default, you will have a large command window, which is where commands are entered and run, a file browser, a workspace window displaying the variables in the current scope, a command history, and beginning with version 4.4, a variable editor.

[1]We will make use of the Octave Forge *symbolic package* where we find it particularly helpful, but the core Octave program, and the focus of this book, is numeric.

1.2 Navigating the GUI

1.2.1 Command history

In Octave, you can save variables that you defined in your session, but this does not save the commands you used, or a whole worksheet. Octave does have a command history that persists between sessions, so past commands can be brought up using the up arrow key, or using the command history list in the GUI.

If you want to save a series of commands that can be reopened, edited, and run again, you can create an Octave script, also known as an .m-file. This will be described in more detail in Chapter 3 (see Section 3.2.2).

1.2.2 File browser

Within the Octave graphical user interface, you should see your current directory listed near the top left. You can click the folder button to navigate to a different directory, such as the desktop, a flash drive, or a folder dedicated to Octave projects. The default start-up directory (and many other options) can be modified in the Octave start-up file `.octaverc`, or the MATLAB-compatible equivalent `startup.m`.

1.2.3 Workspace

Under the file menu, the option "save workspace as" will allow you to save *all* variables in the current scope. The workspace panel lists the current variables. An individual variable can be saved from the variable editor, described below. You can use the "load workspace" option under the file menu to load previously saved variables. Alternately, a variable or workspace file can be loaded by double-clicking on its name in the file browser.

Another approach is to use the manual save and load commands at the command line. If you type save FILENAME var1 var2 ..., Octave will save the specified variables in the file `FILENAME`. If you do not supply a list of variables, then all variables in the current scope will be saved. You can then reload the saved variable(s) at another time by navigating to the appropriate directory and using load FILENAME.

1.2.4 The variable editor

The variable editor allows displaying or editing a variable in a simple spreadsheet format. To use it, double click on the name of the variable in the workspace panel. You can undock the variable editor and maximize it is as a standalone window to facilitate working with larger arrays. If you want to enter data in a variable that does not already exist, you will need to preallocate a matrix of the correct size, for example using the command A = zeros(m, n) to create an $m \times n$ matrix of zeros. You can then open this in the variable editor and enter the data.

1.2.5 Getting help

The full Octave software manual is accessible by changing to the documentation tab at the bottom of the screen, or the shell command help can be used at the Octave prompt. In particular, if you know the name of the command you want to use, help NAME will give the correct syntax. A basic command glossary is available in Appendix B.

Besides the present volume, here are two good free resources to help you get started quickly:

- Simple Examples (from the Octave Manual [3]):
  https://octave.org/doc/v4.0.1/Simple-Examples.html
- Wikibooks Tutorial:
  https://en.wikibooks.org/wiki/Octave_Programming_Tutorial

Additional help can be found with internet searches. Depending on what you are looking for, searches for Octave commands and searches for MATLAB commands can both be useful. Numerous commercial user's guides and textbooks for Octave and/or MATLAB are available. Linear algebra textbooks sometimes contain MATLAB code examples and these generally work in Octave as well.

1.3 Matrices and vectors

1.3.1 Basic arithmetic

The best way to get started is to try some simple problems. Use the following examples as a tutorial to learn your way around the program. Octave knows basic arithmetic and uses the standard order of operations. Try some simple computations:

```
>> 6/2 + 3*(7 - 4)^2
ans = 30
```

Octave ignores white space, so 6/2 and 6 / 2 are interpreted the same way. You can't take shortcuts and leave out implied operations, though. For example, 3(7 − 4) will give an error. Use 3∗(7 − 4).

Vectors and matrices are basic variable types, so it is easier to learn Octave syntax if you already know a little linear algebra. Try this example to enter a row vector and name it **u**. You do not need to enter the comments (indicated by the % sign).

```
>> u = [1  -4  6]    % row vector
u =                  % variable name

   1   -4    6       % output
```

The code u = ... assigns the result of the operation that follows to the variable **u**, which can then be recalled and used in further calculations.

To create a column vector instead, use semicolons:

```
>> u = [1; -4; 6]    % column vector
u =                  % variable name

   1                 % output
  -4
   6
```

Notice that the function of the semicolon is to begin a new row. The same basic syntax is used to enter matrices. For example, here is how to define a 3×3 matrix:

```
>> A = [1 2 -3; 2 4 0; 1 1 1]    % matrix
A =                              % variable name

   1   2  -3                     % output
   2   4   0
   1   1   1
```

In Octave, all of the above variables are really just matrices of different dimensions. A scalar is essentially a 1×1 matrix. Similarly, a row vector is a $1 \times n$ matrix and a column vector is an $m \times 1$ matrix. In the following sections we will take a closer look at the nuances of vector and matrix operations.

1.3.2 Vector operations

We'll start with some simple examples. First, re-enter the column vector **u** from above, if it is not already in memory.

```
>> u = [1; -4; 6]
```

Now enter another column vector **v** and try the following vector operations which illustrate linear combinations, dot product, cross product, and norm (length).

```
>> v = [2; 1; -1]
v =

   2
   1
  -1

>> 2*v + 3*u          % linear combination
ans =

   7
 -10
  16

>> dot(u, v)          % dot product
ans = -8

>> cross(u, v)        % cross product
```

```
ans =

  -2
  13
   9

>> norm(u)          % length of vector u
ans =  7.2801
```

Try a few more operations:

- Find cross(v, u). How does that compare to $\mathbf{u} \times \mathbf{v}$?
- Calculate the length of $\mathbf{v}$, $||\mathbf{v}||$, using norm(v).
- Normalize $\mathbf{v}$ by calculating $\mathbf{v}/||\mathbf{v}||$. This gives a unit vector that points in the same direction as $\mathbf{v}$.

Now let's try a more complicated vector geometry problem, to see some of Octave's potential.

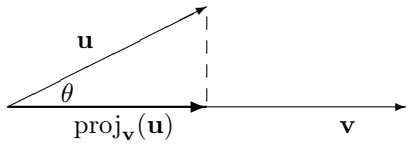


Figure 1.2: Vector projection

The *projection of* $\mathbf{u}$ *onto* $\mathbf{v}$, denoted $\text{proj}_{\mathbf{v}}(\mathbf{u})$, is the component of $\mathbf{u}$ that points in the direction of $\mathbf{v}$. This can be thought of as the shadow $\mathbf{u}$ casts onto $\mathbf{v}$ from a direction orthogonal to $\mathbf{v}$, as shown in Figure 1.2. To find the magnitude of the projection, use basic right-triangle trigonometry:

$$\|\text{proj}_{\mathbf{v}}(\mathbf{u})\| = \|\mathbf{u}\| \cos(\theta)$$

Then, since $\mathbf{u} \cdot \mathbf{v} = \|\mathbf{u}\|\|\mathbf{v}\| \cos(\theta)$,

$$\begin{aligned} \|\text{proj}_{\mathbf{v}}(\mathbf{u})\| &= \|\mathbf{u}\| \cos(\theta) \\ &= \|\mathbf{u}\| \frac{\mathbf{u}\cdot\mathbf{v}}{\|\mathbf{u}\|\|\mathbf{v}\|} \\ &= \frac{\mathbf{u}\cdot\mathbf{v}}{\|\mathbf{v}\|} \end{aligned}$$

This is known as the *scalar projection* of $\mathbf{u}$ onto $\mathbf{v}$. The *vector projection* onto $\mathbf{v}$ is obtained by multiplying the scalar projection by a unit vector that points in the direction of $\mathbf{v}$. Thus,

$$\begin{aligned} \text{proj}_{\mathbf{v}}(\mathbf{u}) &= \frac{\mathbf{u}\cdot\mathbf{v}}{\|\mathbf{v}\|} \frac{\mathbf{v}}{\|\mathbf{v}\|} \\ &= \frac{\mathbf{u}\cdot\mathbf{v}}{\|\mathbf{v}\|^2} (\mathbf{v}) \end{aligned}$$

Since $\mathbf{v} \cdot \mathbf{v} = \|\mathbf{v}\|^2$, this can also be written as:

$$\text{proj}_{\mathbf{v}}(\mathbf{u}) = \frac{\mathbf{u} \cdot \mathbf{v}}{\mathbf{v} \cdot \mathbf{v}} (\mathbf{v})$$

The operations needed for vector projection are easily carried out in Octave. As an example, we will find the projection of $\mathbf{u} = \langle 3, 5\rangle$ onto $\mathbf{v} = \langle 7, 2\rangle$.

```
>> u = [3  5]
u =

   3   5

>> v = [7  2]
v =

   7   2

>> dot(u,v)/(norm(v))^2*v
ans =

   4.0943   1.1698
```

Thus $\text{proj}_{\mathbf{v}}(\mathbf{u}) = \langle 4.0943, 1.1698\rangle$. Later, in Example 5.5.2, we will see how to create our own user-defined function to automate the above steps.

1.3.3 Matrix operations

Matrix operations are simple and intuitive in Octave. We'll start with multiplication.

Let $A = \begin{bmatrix} 1 & 2 & -3 \\ 2 & 4 & 0 \\ 1 & 1 & 1 \end{bmatrix}$ and $B = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 0 & -2 & -4 & 6 \\ 1 & -1 & 0 & 0 \end{bmatrix}$. Find AB.

```
>> A = [1  2  -3; 2  4  0; 1  1  1]    % matrix
A =                                     % variable name

   1   2  -3                           % output
   2   4   0
   1   1   1

>> B = [1  2  3  4; 0  -2  -4  6; 1  -1  0  0]
B =

   1   2   3   4
   0  -2  -4   6
   1  -1   0   0

>> A*B                                  % multiply A and B
ans =                                   % result stored as 'ans'

  -2   1  -5  16                        % answer
   2  -4 -10  32
   2  -1  -1  10
```

Notice that the result is stored in the temporary variable ans.

Arithmetic operations in Octave are always assumed to be matrix operations, unless otherwise specified (see Section 1.4.1). Therefore, for A and B defined as above, we can compute things like $4A$ or AB by entering, respectively, 4*A or A*B, but operations like B*A or A+B throw errors (why?).

To get the transpose of a matrix, use the single quote[2]. For example, try calculating $B^T A$.

```
>> B'*A          % B' is the transpose of B
ans =

    2     3    -2
   -3    -5    -7
   -5   -10    -9
   16    32   -12
```

To perform basic matrix arithmetic, we also need identity matrices, which are easy to generate with the eye(n) command, where n is the dimension of the matrix. Let's find $2A - 4I$.

```
>> 2*A - 4*eye(3)        % eye(3) is a 3x3 identity matrix
ans =

   -2    4   -6
    4    4    0
    2    2   -2
```

Octave can also find determinants, inverses, and eigenvalues. For example, try these commands.

```
>> det(A)                              % determinant
ans = 6

>> inv(A)                              % matrix inverse
ans =

    0.66667   -0.83333    2.00000
    0.33333    0.66667   -1.00000
    0.33333    0.16667    0.00000

>> eig(A)                              % eigenvalues
ans =

   4.52510 + 0.00000i
   0.73745 + 0.88437i
   0.73745 - 0.88437i
```

Notice that our matrix has one real and two complex eigenvalues. Octave handles complex numbers, of course! Eigenvalues will be discussed in more detail in Chapter 5. Octave can also compute many other matrix values, such as rank:

```
>> rank(A)     % matrix rank
ans =  3
```

[2]Technically this gives the *conjugate transpose*, but for a matrix with real entries, there is no difference between the conjugate transpose and the ordinary transpose.

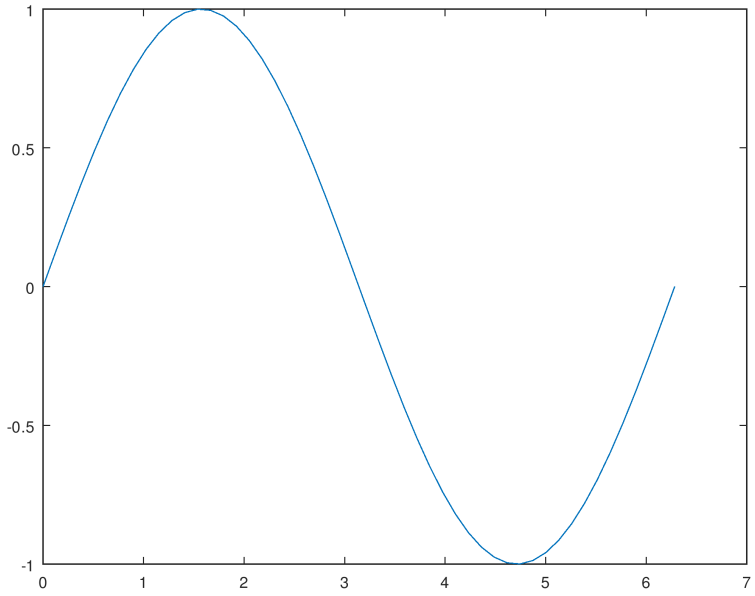


Figure 1.3: Default graph of $y = \sin(x)$ on $[0, 2\pi]$

1.4 Plotting

Basic two-dimensional plotting of functions in Octave is accomplished by creating a vector for the independent variable and a second vector for the range of the function. There are several forms for the syntax and we will attempt to outline the simplest methods here[3].

Let's start by plotting the graph of the function $\sin(x)$ on the interval $[0, 2\pi]$ (Figure 1.3). Like a typical graphing calculator, Octave will simply plot a series of points and connect the dots to represent the curve. The process is less automated in Octave (but in the end, much more powerful). We begin by creating a vector of x-values.

```
>> x = linspace(0, 2*pi, 50);
```

Notice the format linspace(start_val , end_val, n). This creates a row vector of 50 evenly spaced values beginning at 0 and going up to 2π. The smaller the increment, the smoother the curve will look. In this case, 50 points should be suitable. The semicolon at the end of the line is to suppress the output to the screen, since we don't need to see all the values in the vector. Now, we want to create a vector of the corresponding y-values. Use this command:

```
>> y = sin(x);
```

Now, to plot the function, use the plot command:

```
>> plot(x, y);
```

[3]See also http://www.gnu.org/software/octave/doc/interpreter/Plotting.html and http://en.wikibooks.org/wiki/Octave_Programming_Tutorial/Plotting

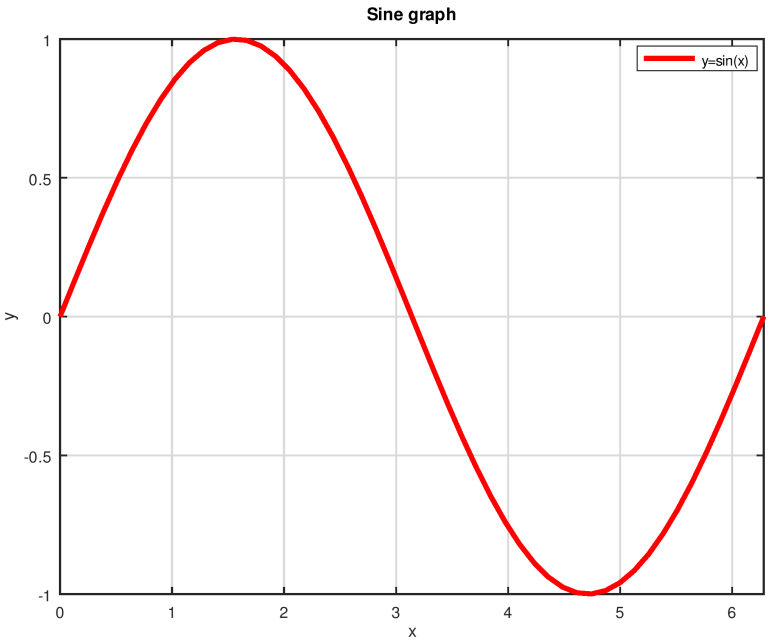


Figure 1.4: Improved graph of $y = \sin(x)$ on $[0, 2\pi]$

You should see the graph of $f(x) = \sin(x)$ pop up in a new window.

Figure 1.3 shows the default graph. You may wish to customize it a little bit. For example, the x-axis extends too far. We can set the window with the axis command. The window is controlled by a vector of the form [Xmin Xmax Ymin Ymax]. Let's set the axes to match the domain and range of the function.

```
>> axis([0 2*pi -1 1]);
```

We may want to change the color (to, say, red) or make the line thicker. We can add a grid to help guide our eye. In addition, a graph should usually be labeled with a title, axis labels, and legend. Try these options to get the improved graph shown in Figure 1.4.

```
>> plot(x, y, 'r', 'linewidth', 3)
>> grid on
>> xlabel('x');
>> ylabel('y');
>> title('Sine graph');
>> legend('y=sin(x)');
```

Note that some adjustments, like zooming in, or turning on the grid, can be done within the graph window using the controls provided. Some standard color options are red, green, blue, cyan, and magenta, which can be specified with their first letter in single quotes.

Now, let's try plotting points. The procedure is essentially the same, but we use an option to specify the marker we want. Some marker options are o, +, or *. We will plot the set of points $\{(1, 1), (2, 2), (3, 5), (4, 4)\}$ using circles as our marker. First, clear the variables from

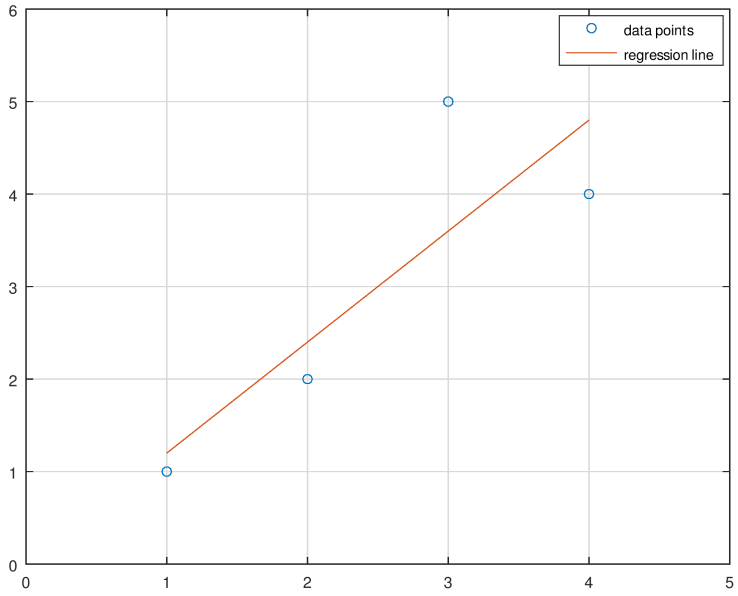


Figure 1.5: Scatter plot with regression line

the workspace and clear any existing graphs. Then define a vector of x-values and a vector of y-values and use the plot command.

```
>> clear; clf;
>> x = [1 2 3 4]
>> y = [1 2 5 4]
>> plot(x, y, 'o')
```

Now suppose we want to graph the line $y = 1.2x$ on the same set of axes (this is the line of best fit for this data). To add to our current graph we need to use the command hold on. Then any new plots will be drawn onto the current axes. We can switch back later with hold off.

```
>> hold on
>> plot(x, 1.2*x)
```

Now we should see four points and the graph of the line. Alternately, we can create multiple plots within a single plot command. Try this, for example:

```
>> clear; clf;
>> x = [1 2 3 4];
>> y1 = [1 2 5 4];
>> y2 = 1.2*x;
>> plot(x, y1, 'o', x, y2)
>> axis([0 5 0 6]);
>> grid on;
>> legend('data points', 'regression line');
```

Notice that sets of input and output variables come in pairs, followed by any options that apply to that pair. The result is shown in Figure 1.5.

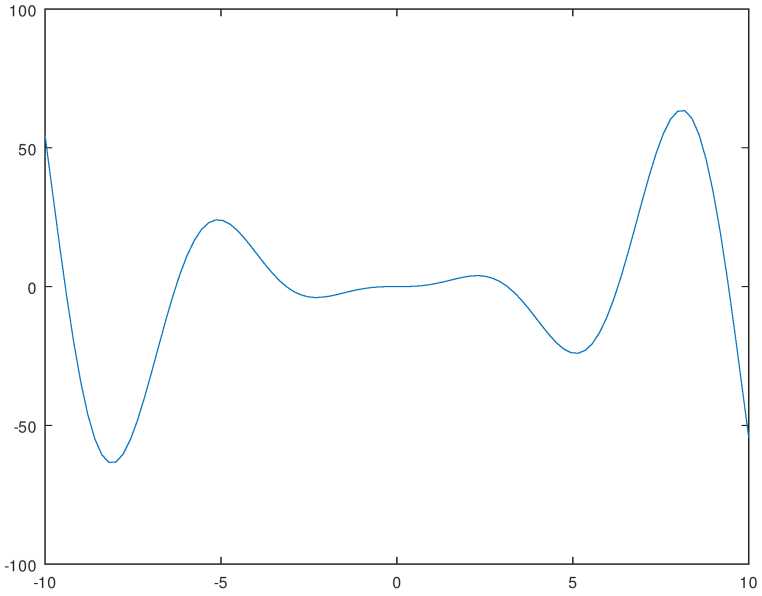


Figure 1.6: Graph of $y = x^2 \sin(x)$

1.4.1 Elementwise operations

An important consideration when working with a more complex function like, say, $y = x^2 \sin(x)$, is that Octave will regard the product and exponent as matrix operations, unless we indicate otherwise. The same is true for division. To avoid errors when we are evaluating a function at a numeric input vector, we need to use the *elementwise* versions of exponentiation, multiplication, and division between variables. This is done by preceding the operation in question with a period, as in .^ or .* or ./.

These commands are incorrect and will cause errors:

```
>> x = linspace(-10, 10, 100);
>> plot(x, x^2*sin(x))  % incorrect syntax: not elementwise operations
error: for A^b, A must be a square matrix. Use .^ for elementwise power.
error: evaluating argument list element number 2
```

But this will do the trick:

```
>> x = linspace(-10, 10, 100);
>> plot(x, x.^2.*sin(x))  % correct: elementwise exponent and product
```

The result is shown in Figure 1.6.

It is important to remember to use elementwise multiplication, exponentiation, and division, except when you are actually intending to execute a matrix operation. Failing to do so is the source of many errors and considerable frustration for beginning Octave users.

1.4.2 Plot options

The following table summarizes some standard options that can be used with the plot command.

PLOT OPTIONS

MARKER	'+'	crosshair	COLOR	'k'	black
	'o'	circle		'r'	red
	'*'	star		'g'	green
	'.'	point		'b'	blue
	's'	square		'm'	magenta
	'^'	triangle		'c'	cyan
SIZE	'linewidth', n (*n* is some positive value)				
	'markersize', n				
LINE STYLE	'−'	solid line (default)			
	'−−'	dashed line			
	':'	dotted line			

Several options may be combined. For example, plot(x, y, 'ro:') indicates red color with circle markers joined by dotted lines.

Here are several key functions for providing textual labels:

PLOT LABELS

HORIZONTAL AXIS LABEL	xlabel('axis name');
VERTICAL AXIS LABEL	ylabel('axis name');
LEGEND	legend('curve 1', 'curve 2', ...) ;
TITLE	title ('plot title ');

The position of the legend may be modified using the command legend('location', 'position'), where "position" is a string giving a compass position, like northeast (which is the default), north, south, east, southwest, etc. Type help legend for a full list of options.

1.4.3 Saving plots

If we have created a good plot, we probably want to save it. The easiest option is to use copy and paste from the plot window. You can also use the "save as" option under the file menu to save the plot in various image formats.

An alternate method is to save the plot directly by "printing" it to a file. Octave supports several image formats. In the example below, the PNG format is used. To save the current graph as a PNG, use this syntax:

```
>> print filename.png -dpng
```

Here `filename.png` is whatever file name you want (include the extension). You can replace PNG with other image formats, such as JPG or EPS. The file will be saved in your current working directory.

Chapter 1 Exercises

Begin each problem with no variables stored. You can clear any previous results with the command clear.

1. For practice saving and loading variables, try the following.
   (a) Create a new directory called "octave_projects".
   (b) Change to the octave_projects directory.
   (c) Save the example matrices A and B from above in a text file named `matrices.mat`.
   (d) Quit Octave.
   (e) Restart Octave and reload the saved matrices.

2. Let $\mathbf{u} = \langle 2, -4, 0\rangle$ and $\mathbf{v} = \langle 3, 1.5, -7\rangle$. Find each of the following.
   (a) $\mathbf{w} = 2\mathbf{u} + 5\mathbf{v}$
   (b) $d = \mathbf{u} \cdot \mathbf{v}$
   (c) $l = ||\mathbf{u}||$
   (d) $\mathbf{u_1}$ = a unit vector that points in the direction of $\mathbf{u}$
   (e) $\mathbf{n}$ = a vector orthogonal to both $\mathbf{u}$ and $\mathbf{v}$
   (f) $\mathbf{p} = \text{proj}_{\mathbf{v}}(\mathbf{u})$

   Be sure to use the variable names indicated to store your answers. Save your workspace including all of the required variables. What does the dot product reveal about $\mathbf{u}$ and $\mathbf{v}$? How did you produce a vector mutually orthogonal to $\mathbf{u}$ and $\mathbf{v}$?

3. Enter the following matrices.

$$A = \begin{bmatrix} 1 & -3 & 5 \\ 2 & -4 & 3 \\ 0 & 1 & -1 \end{bmatrix}, B = \begin{bmatrix} 1 & -1 & 0 & 0 \\ -3 & 0 & 7 & -6 \\ 2 & 1 & -2 & -1 \end{bmatrix}, \text{ and } I_3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

   Use Octave to compute each of the following.
   (a) $d = \det(A)$
   (b) $r = \text{rank}(B)$
   (c) $C = 2A + 4I$
   (d) $D = A^{-1}$
   (e) $E = AB$
   (f) $F = B^T A^T$

   Use the variable names indicated to store your answers. Save your workspace including all of the required variables. Did you notice anything about AB and $B^T A^T$? If so, explain the relationship between these quantities.

4. Plot the following set of points, using triangle markers:

$$\{(-1,5),(1,4),(2,2.5),(3,0)\}$$

Turn on the grid. Give the plot the title "Scatter plot" and save it as a PNG or JPG image file.

5. Modify the plot of $y = x^2\sin(x)$ given in Figure 1.6 as follows:

   (a) Make the graph of $y = x^2\sin(x)$ a thick red line.

   (b) Graph $y = x^2$ and $y = -x^2$ on the same axes, as thin black dotted lines.

   (c) Use a legend to identify each curve.

   (d) Add a title.

   (e) Add a grid.

   Save the plot as a PNG or JPG image file.

6. Let $f(x) = 2 - e^{x-1}$.

   (a) Plot a graph of the function as a thick blue line on $[-3,3]\times[-5,5]$. Note that exp(x) can be used for the exponential function.

   (b) Add the function's horizontal asymptote to your figure as a dashed black line. The ones command provides a useful trick for the required constant term:

```
>> % plot constant function y = k
>> plot(x, k*ones(size(x)))
```

   (c) Add a graph of the "parent function," $y = e^x$, on the same axes as a thin red line.

   (d) Add a legend to your figure identifying the three curves as "transformed exponential function," "horizontal asymptote," and "parent function."

   (e) You have probably noticed that our graphs do not show the traditional x- and y-coordinate axes—and usually this is not a problem. But, for this graph we will add them. Here is one method that can be used:

```
>> % plot x- and y-coordinate axes as thin black lines
>> % [a b] = horizontal axis limits
>> % [c d] = vertical axis limits
>> plot([a b], [0 0], 'k', [0 0], [c d], 'k')
```

   With the coordinate axes plotted manually, we may now wish to turn off the default box-format axes:

```
>> axis off
```

   (f) Showing the coordinate axes makes the intercepts more clearly apparent. Highlight them by adding circle markers at the x- and y-intercepts. Use the text function to label these points with their coordinates. This is the syntax to use:

```
>> % add a text label to plot at coordinates (x, y)
>> text(x, y, 'label')
```

Save the plot as a PNG or JPG image file.

7. The load command described in Section 1.2.3 works well for loading data that is in Octave format. But sometimes it is necessary to import data that is not already in Octave format. For data in a spreadsheet document (or that can be copied into a spreadsheet to manipulate), the simplest option is to save your data as a CSV file (comma-separated values) and load it in Octave using csvread('filename.csv'). This really only works for purely numeric data, so if necessary, remove any special formatting and strip out text headings and labels. For practice with this useful method of importing data, try the following exercise:

   Enter the data matrix shown below in a spreadsheet program, like LibreOffice Calc[4] (or Excel):

	A	B	C
1	1	2	3
2	-1	0	0.2
3	=A1+A2	=pi()	=7/3

   The formulas on line 3 should evaluate to decimal values. Save the resulting data as `example1.csv`. Navigate to the appropriate directory and load the example matrix in Octave using this command:

   ```
   >> A = csvread('example1.csv')
   ```

   Then, save A as an Octave-format data file, `example1.mat`.

[4] https://www.libreoffice.org/

Chapter 2

Matrices and linear systems

Octave is a powerful tool for many problems in linear algebra. We have already seen some of the basics in Section 1.3. In this chapter, we will consider systems of linear equations, polynomial curve fitting, and matrix transformations.

2.1 Linear systems

2.1.1 Gaussian elimination

Octave has sophisticated algorithms built in for solving systems of linear equations, but it is useful to start with the more basic process of Gaussian elimination. Using Octave for Gaussian elimination lets us practice the procedure, without the inevitable arithmetic errors that come when doing elimination by hand. It also teaches useful Octave syntax and methods for manipulating matrices.

Row operations are easy to carry out. But first, we need to see how matrices and vectors are indexed in Octave. Consider the following matrix.

```
>> B = [1  2  3  4; 0 -2 -4  6; 1 -1  0  0]
B =

   1   2   3   4
   0  -2  -4   6
   1  -1   0   0
```

If we enter B(2, 3), then the value returned is -4. This is the scalar stored in row 2, column 3. We can also pull out an entire row vector or column vector using the colon operator. A colon can be used to specify a limited range, or if no starting or ending value is specified, it gives the full range. For example, B(1, :) will give every entry out of the first row.

```
>> B(1, :)  % from matrix B: row 1, all columns
ans =
```

1 2 3 4

Now, let's use this notation to carry out basic row operations on B to reach row echelon form.

Example 2.1.1. Consider the system of linear equations represented by the augmented matrix

$$B = \left[\begin{array}{ccc|c} 1 & 2 & 3 & 4 \\ 0 & -2 & -4 & 6 \\ 1 & -1 & 0 & 0 \end{array}\right]$$

Use row operations to put B into row echelon form, then solve by backward substitution. Compare to the row-reduced echelon form computed by Octave.

Solution. The first operation is to replace row 3 with -1 times row 1, added to row 3.

```
>> % new row 3 = -1*row 1 + row 3
>> B(3, :) = (-1)*B(1, :) + B(3, :)
ans =

   1   2   3   4
   0  -2  -4   6
   0  -3  -3  -4
```

Next, we will replace row 3 with -1.5 times row 2, added to row 3.

```
>> % new row 3 = -1.5*row 2 + row 3
>> B(3, :) = -1.5*B(2, :) + B(3, :)
ans =

   1   2   3    4
   0  -2  -4    6
   0   0   3  -13
```

The matrix is now in row echelon form. We could continue using row operations to reach row-reduced echelon form, but it is more efficient to simply write out the corresponding linear system on paper and solve by backward substitution. The solution vector is $\langle \frac{17}{3}, \frac{17}{3}, -\frac{13}{3} \rangle$.

Octave also has a built-in command, rref, to find the row-reduced echelon form of the matrix directly.

```
>> rref(B)
ans =

   1.00000   0.00000   0.00000   5.66667
   0.00000   1.00000   0.00000   5.66667
   0.00000   0.00000   1.00000  -4.33333
```

From here, the solution to the system is evident. Notice that everything is now expressed as floating point numbers (i.e., decimals). Five decimal places are displayed by default. The variables are actually stored with higher precision and it is possible to display more decimal places, if desired (type: format long). □

2.1.2 Left division

The built-in operation for solving linear systems of the form $A\mathbf{x} = \mathbf{b}$ in Octave is called *left division* and is entered as A\b. This is "conceptually equivalent" to the product $A^{-1}\mathbf{b}$ ([3]). Let's try the left division operation on the system from the prior example, with augmented matrix B.

Example 2.1.2. Use left division to solve the system of equations with augmented matrix B.

$$B = \left[\begin{array}{ccc|c} 1 & 2 & 3 & 4 \\ 0 & -2 & -4 & 6 \\ 1 & -1 & 0 & 0 \end{array}\right]$$

Solution. To use left division, we need to extract the coefficient matrix and vector of right-side constants. Let's call the coefficient matrix A and the right-side constants $\mathbf{b}$. (You have probably already noticed that Octave is case-sensitive.)

```
>> B = [1 2 3 4; 0 -2 -4 6; 1 -1 0 0]  % re-enter B, if necessary
B =

   1   2   3   4
   0  -2  -4   6
   1  -1   0   0

>> A = B(:, 1:3) % extract coefficient matrix
A =

   1   2   3
   0  -2  -4
   1  -1   0

>> b = B(:, 4)   % extract right side constants
b =

   4
   6
   0

>> A\b           % solve system Ax = b
ans =

   5.6667
   5.6667
  -4.3333
```

The solution vector matches what we found by Gaussian elimination. □

2.1.3 LU decomposition

LU decomposition is a matrix factorization that encodes the results of the Gaussian elimination algorithm. The goal is to write $A = LU$, where L is a unit lower triangular matrix and U is an

upper triangular matrix. We will see that this factored form can be used to easily solve $A\mathbf{x} = \mathbf{b}$.

The process is best explained with an example. We will not attempt to justify why the algorithm works; refer to [6] for the underlying theory.

Example 2.1.3. Find an LU decomposition for

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 0 & -2 & -4 \\ 1 & -1 & 0 \end{bmatrix}$$

Solution. This is the same coefficient matrix we row-reduced in Example 2.1.1. We proceed the same way, carefully noting the multiplier used to obtain each 0. The lower triangular L starts as an identity matrix, then the negative of each multiplier used in the elimination process is placed into the corresponding entry of L.

The first zero in position $(2, 1)$ is already there, so we put 0 for that multiplier in the corresponding position of L. Then we replace row 3 with -1 times row 1 plus row 3. The negative of this multiplier is $-(-1) = 1$, which is entered in L at the point where the 0 was obtained.

At this point, we have two entries for L along with a partly reduced A:

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 0 & -2 & -4 \\ 1 & -1 & 0 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 2 & 3 \\ 0 & -2 & -4 \\ 0 & -3 & -3 \end{bmatrix}; L = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & ? & 1 \end{bmatrix}$$

The next step is to replace row 3 using -1.5 times row 2. Thus we put $-(-1.5) = 1.5$ in the corresponding position of L. Once A has reached row echelon form, we have the desired upper triangular matrix U.

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 0 & -2 & -4 \\ 1 & -1 & 0 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 2 & 3 \\ 0 & -2 & -4 \\ 0 & -3 & -3 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 2 & 3 \\ 0 & -2 & -4 \\ 0 & 0 & 3 \end{bmatrix} = U$$

$$L = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 1.5 & 1 \end{bmatrix}, U = \begin{bmatrix} 1 & 2 & 3 \\ 0 & -2 & -4 \\ 0 & 0 & 3 \end{bmatrix}$$

So, to review, U is the row echelon form of A and L is an identity matrix with the negatives of the Gaussian elimination multipliers placed into the corresponding positions where they were used to obtain zeros.

Let's check to see if it worked.

```
>> L = [1 0 0; 0 1 0; 1 1.5 1]
L =

    1.00000    0.00000    0.00000
    0.00000    1.00000    0.00000
    1.00000    1.50000    1.00000
```

```
>> U = [1  2  3; 0 -2 -4; 0 0 3]
U =

    1    2    3
    0   -2   -4
    0    0    3

>> L*U
ans =

    1    2    3
    0   -2   -4
    1   -1    0
```

It worked! In fact, the procedure outlined in this example will work anytime Gaussian elimination can be performed *without row interchanges*. □

Now, let's see how the LU form can be used to solve linear systems $A\mathbf{x} = \mathbf{b}$. If $A = LU$, then the system $A\mathbf{x} = \mathbf{b}$ can be written as $LU\mathbf{x} = \mathbf{b}$. Let $U\mathbf{x} = \mathbf{y}$. Then we can proceed in two steps:

1. Solve $L\mathbf{y} = \mathbf{b}$.
2. Solve $U\mathbf{x} = \mathbf{y}$.

Since we are dealing with triangular matrices, each step is easy.

Example 2.1.4. Solve $A\mathbf{x} = \mathbf{b}$, where $A = \begin{bmatrix} 1 & 2 & 3 \\ 0 & -2 & -4 \\ 1 & -1 & 0 \end{bmatrix}$ and $\mathbf{b} = \begin{bmatrix} 4 \\ 6 \\ 0 \end{bmatrix}$, using LU decomposition.

Solution. We already have the LU decomposition. Since $L = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 1.5 & 1 \end{bmatrix}$, the first step is to solve:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 1.5 & 1 \end{bmatrix} \cdot \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} 4 \\ 6 \\ 0 \end{bmatrix}$$

The corresponding systems of equations is

$$\begin{array}{rcrcrcl} y_1 & & & & & = & 4 \\ & & y_2 & & & = & 6 \\ y_1 & + & 1.5y_2 & + & y_3 & = & 0 \end{array}$$

Starting with the first row and working down, this system is easily solved by *forward substitution*. We can see that $y_1 = 4$ and $y_2 = 6$. Substituting these values into the third equation and solving for y_3 gives $y_3 = -13$. Thus the intermediate solution for $\mathbf{y}$ is $\begin{bmatrix} 4 \\ 6 \\ -13 \end{bmatrix}$.

Step two is to solve $U\mathbf{x} = \mathbf{y}$, which looks like:

$$\begin{bmatrix} 1 & 2 & 3 \\ 0 & -2 & -4 \\ 0 & 0 & 3 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 4 \\ 6 \\ -13 \end{bmatrix}$$

This is easily solved by backward substitution to get $\mathbf{x} = \begin{bmatrix} 17/3 \\ 17/3 \\ -13/3 \end{bmatrix}$. □

If row interchanges are used, then A is multiplied by a *permutation matrix* and the decomposition takes the form $PA = LU$. This is the default form of the LU decomposition given by Octave using the command [L U P] = lu(A).

Example 2.1.5. Find an LU decomposition (with permutation) for

$$A = \begin{bmatrix} -7 & -2 & 9 & 4 \\ -4 & -9 & 3 & 0 \\ -3 & 4 & 6 & -2 \\ 6 & 7 & -4 & -8 \end{bmatrix}$$

Solution. We will use Octave for this.

```
>> A = [-7 -2 9 4; -4 -9 3 0; -3 4 6 -2; 6 7 -4 -8]
A =

   -7   -2    9    4
   -4   -9    3    0
   -3    4    6   -2
    6    7   -4   -8

>> [L U P] = lu(A)
L =

    1.00000    0.00000    0.00000    0.00000
    0.57143    1.00000    0.00000    0.00000
   -0.85714   -0.67273    1.00000    0.00000
    0.42857   -0.61818    0.36000    1.00000

U =

   -7.00000   -2.00000    9.00000    4.00000
    0.00000   -7.85714   -2.14286   -2.28571
    0.00000    0.00000    2.27273   -6.10909
    0.00000    0.00000    0.00000   -2.92800

P =

Permutation Matrix

    1    0    0    0
    0    1    0    0
```

$$\begin{matrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{matrix}$$

Then we have the factorization $PA = LU$, where PA is a row-permutation of A. □

Refer to Exercise 4 to see how $PA = LU$ can be used to solve a linear system, using a method almost to identical to what we did in Example 2.1.4.

LU decomposition is widely used in numerical linear algebra. In fact, it is the basis of how Octave's left division operation works. It is especially efficient to use LU decomposition when one is solving several systems of equations that all have the same coefficient matrix, but different right side constants. The LU decomposition only needs to be done once for all of the systems with that coefficient matrix.

2.2 Polynomial curve fitting

In statistics, the problem of fitting a straight line to a set of data is often considered. We tackle the more general problem of fitting a polynomial to a set of points.

Example 2.2.1. Set-up and solve the normal equations to find the least-squares parabola for the set of points in the following 6×2 data matrix D.

$$D = \begin{bmatrix} 1 & 1 \\ 2 & 2 \\ 3 & 5 \\ 4 & 4 \\ 5 & 2 \\ 6 & -3 \end{bmatrix}$$

The matrix shows x-values in column 1 and y-values in column 2.

Solution. Enter the data matrix in Octave and extract the x- and y-data to column vectors. Then plot the points to get a sense of what the data look like.

```
>> D = [1  1; 2  2; 3  5; 4  4; 5  2; 6  -3]
>> xdata = D(:, 1)
>> ydata = D(:, 2)
>> plot(xdata, ydata, 'o-')  % plot line segments with circle markers
```

In this case, we are constructing a model of the form $y = ax^2 + bx + c$, but it is easy to see how our approach generalizes to polynomials of any degree (including linear functions). By plugging in the given data to the proposed equation, we obtain the following system of linear equations.

$$\begin{bmatrix} 1 & 1 & 1 \\ 4 & 2 & 1 \\ 9 & 3 & 1 \\ 16 & 4 & 1 \\ 25 & 5 & 1 \\ 36 & 6 & 1 \end{bmatrix} \cdot \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 5 \\ 4 \\ 2 \\ -3 \end{bmatrix}$$

Notice the form of the coefficient matrix, which we'll call A. The third column is all ones, the second column is the x-values, and the first column is the square of the x-values (this column would not appear if we were using a linear model). The corresponding right-side constants are the y-values. There are several ways to construct the coefficient matrix in Octave. One approach is to use the ones command to create a matrix of ones of the appropriate size, and then overwrite the first and second columns with the correct data.

```
>> A = ones(6, 3);
>> A(:, 1) = xdata.^2;
>> A(:, 2) = xdata
A =

     1    1    1
     4    2    1
     9    3    1
    16    4    1
    25    5    1
    36    6    1
```

Note the use of elementwise exponentiation to square each value of the x-data vector. Our system is inconsistent. It can be shown that the least-squares solution comes from solving the *normal equations*, $A^TA\mathbf{p} = A^T\mathbf{y}$, where $\mathbf{p}$ is the vector $\begin{bmatrix} a \\ b \\ c \end{bmatrix}$ of polynomial coefficients.

We can use Octave to construct the normal equations.

```
>> A'*A
ans =

   2275    441     91
    441     91     21
     91     21      6

>> A'*ydata
ans =

   60
   28
   11
```

The corresponding augmented matrix is:

$$\left[\begin{array}{ccc|c} 2275 & 441 & 91 & 60 \\ 441 & 91 & 21 & 28 \\ 91 & 21 & 6 & 11 \end{array}\right]$$

We can then solve the problem using Gaussian elimination. Here is one way to create the augmented matrix and row-reduce it:

```
>> B = A'*A;
>> B(:, 4) = A'*ydata;
```

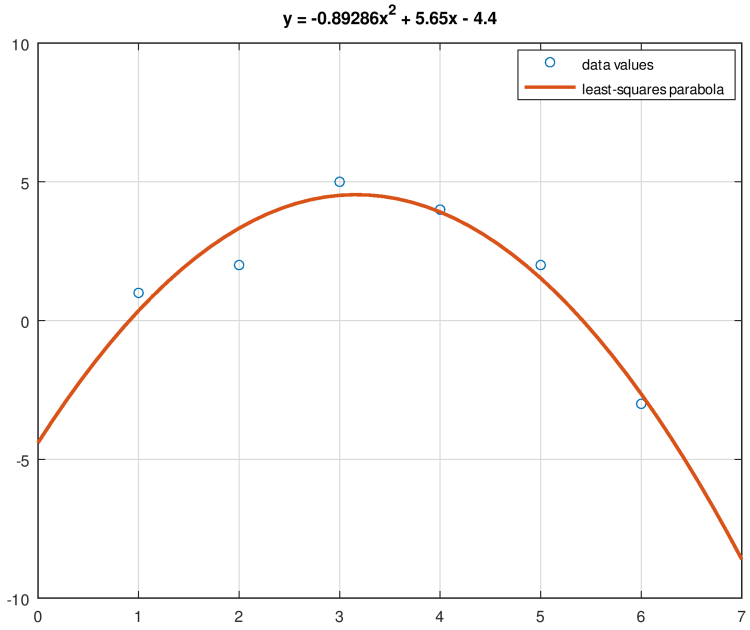


Figure 2.1: Least-squares parabola

```
>> rref(B)
ans =

    1.00000   0.00000   0.00000  -0.89286
    0.00000   1.00000   0.00000   5.65000
    0.00000   0.00000   1.00000  -4.40000
```

Thus the correct quadratic equation is $y = -0.89286x^2 + 5.65x - 4.4$.

Now we plot a graph of this parabola together with our original data points. These are the commands used to create the graph:

```
>> x = linspace(0, 7, 50);
>> y = -0.89286*x.^2 + 5.65*x - 4.4;
>> plot(xdata, ydata, 'o', x, y, 'linewidth', 2)
>> grid on;
>> legend('data values', 'least-squares parabola')
>> title('y = -0.89286x^2 + 5.65x - 4.4')
```

The graph is shown in Figure 2.1. □

You may be wondering if any of this process can be automated by built-in Octave functions. Yes! If we want Octave to do all of the work for us, we can use the built-in function for polynomial fitting, polyfit. The syntax is polyfit(x, y, order), where "order" is the degree of the polynomial desired.

Octave is configured to work with polynomials by associating their coefficients with a simple

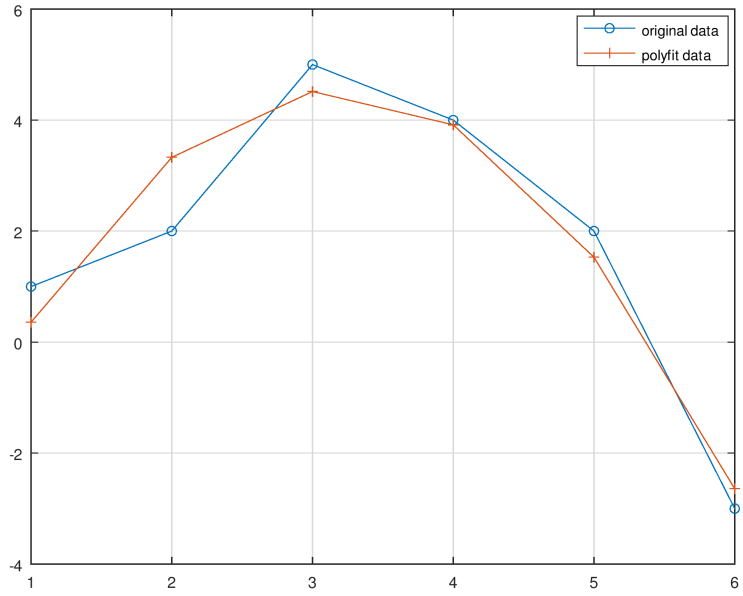


Figure 2.2: Plot of original data vs. polyfit data

row vector. For example, [2 −3 4] corresponds to $2x^2 - 3x + 4$. The polyval function can be used to evaluate an Octave-format polynomial using the syntax polyval(P, x).

Example 2.2.2. Use polyfit to find the least-squares parabola for the following data:

x	1	2	3	4	5	6
y	1	2	5	4	2	-3

Graph the original data and polyfit data together on the same axes.

Solution. This is the same data as in Example 2.2.1. Re-enter the data values if necessary.

We use polyfit to determine the equation, then the polyval function to evaluate the polynomial at the given x-values.

```
>> P = polyfit(xdata, ydata, 2) % degree two polynomial fit
P =

   -0.89286    5.65000   -4.40000

>> y = polyval(P, xdata); % evaluate polynomial P at input xdata
>> plot(xdata, ydata, 'o-', xdata, y, '+-');
>> grid on;
>> legend('original data', 'polyfit data');
```

The graph is shown in Figure 2.2. □

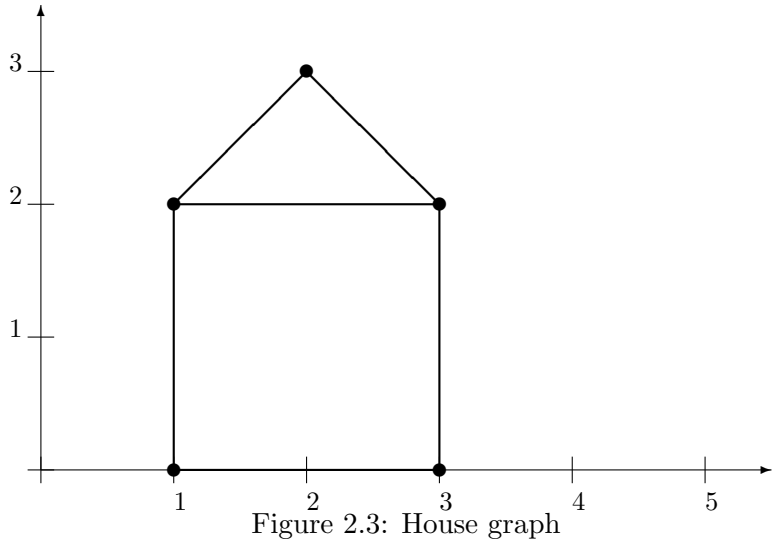


Figure 2.3: House graph

2.3 Matrix transformations

Matrices and matrix transformations play a key role in computer graphics. There are several ways to represent an image as a matrix. The approach we take here is to list a series of vertices that are connected sequentially to produce the edges of a simple graph. We write this as a $2 \times n$ matrix where each column represents a point in the figure. As a simple example, let's try to encode a "house graph." First, we draw the figure on a grid and record the coordinates of the points, as in Figure 2.3.

There are many ways to encode this in a matrix. An efficient method is to choose a path that traverses each edge exactly once, if possible[1]. Here is one such matrix, starting from $(1, 2)$ and traversing counterclockwise.

$$D = \begin{bmatrix} 1 & 1 & 3 & 3 & 2 & 1 & 3 \\ 2 & 0 & 0 & 2 & 3 & 2 & 2 \end{bmatrix}$$

Try plotting it in Octave and see if it worked.

```
>> D = [1 1 3 3 2 1 3; 2 0 0 2 3 2 2]
D =

   1   1   3   3   2   1   3
   2   0   0   2   3   2   2

>> x = D(1, :);
>> y = D(2, :);
>> plot(x, y);
```

You may want to zoom out to see the origin. Then the graph appears correct.

[1]This is called an *Eulerian path*. Such a path exists if the graph has exactly 0 or 2 vertices with odd degree.

2.3.1 Rotation

Now that we have a representation of a digital image, we consider various ways to transform it. Rotations can be obtained using multiplication by a special matrix.

A rotation of the point (x, y) about the origin is given by

$$R \cdot \begin{bmatrix} x \\ y \end{bmatrix}$$

where

$$R = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}$$

and θ is the angle of rotation (measured counterclockwise).

For example, what happens to the point $(1, 0)$ under a $90°$ rotation?

$$\begin{bmatrix} \cos(90°) & -\sin(90°) \\ \sin(90°) & \cos(90°) \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

The rotation appears to work, at least in this case. Try a few more points to convince yourself. Notice that a rotation about the origin corresponds to moving along a circle, thus the trigonometry is fairly straightforward to work out.

Now, to produce rotations of a data matrix D, encoded as above, we only need to compute the matrix product RD.

Example 2.3.1. Rotate the house graph through $90°$ and $225°$.

Solution. Note that each θ must be converted to radians. Here we go:

```
>> D = [1 1 3 3 2 1 3; 2 0 0 2 3 2 2];
>> x = D(1, :);
>> y = D(2, :);

>> % execute a 90 degree rotation
>> theta1 = 90*pi/180;
>> R1 = [cos(theta1) -sin(theta1); sin(theta1) cos(theta1)];
>> RD1 = R1*D;
>> x1 = RD1(1, :);
>> y1 = RD1(2, :);

>> % execute a 225 degree rotation
>> theta2 = 225*pi/180;
>> R2 = [cos(theta2) -sin(theta2); sin(theta2) cos(theta2)];
>> RD2 = R2*D;
>> x2 = RD2(1, :);
>> y2 = RD2(2, :);

>> % plot original and rotated figures
>> plot(x, y, 'bo-', x1, y1, 'ro-', x2, y2, 'mo-')
```

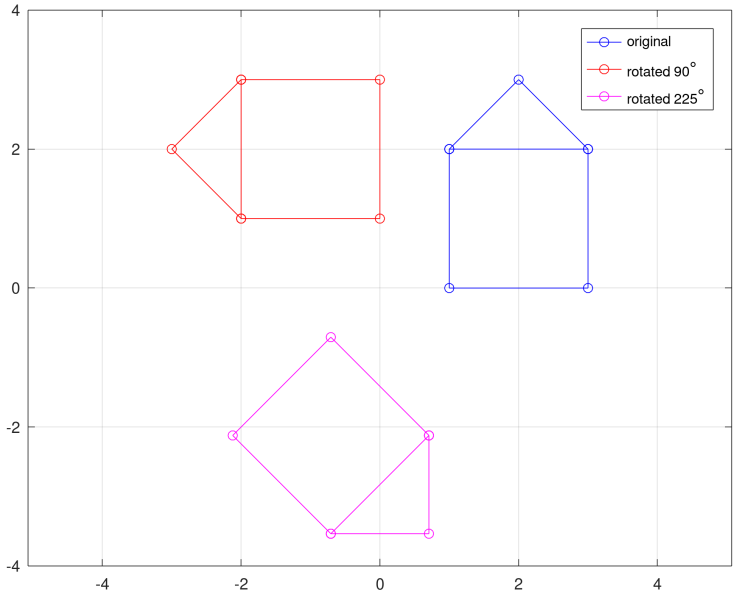


Figure 2.4: Rotations of the house graph

```
>> axis([-4 4 -4 4], 'equal');
>> grid on;
>> legend('original', 'rotated 90 deg', 'rotated 225 deg');
```

Note the combined plot options to set color, marker, and line styles. The original and rotated graphs are shown in Figure 2.4. Notice that the rotation is about the origin. For rotations about an arbitrary point, see Exercise 14. □

2.3.2 Reflection

If ℓ is a line through the origin, then a reflection of the point (x, y) in the line ℓ is given by

$$R \cdot \begin{bmatrix} x \\ y \end{bmatrix}$$

where

$$R = \begin{bmatrix} \cos(2\theta) & \sin(2\theta) \\ \sin(2\theta) & -\cos(2\theta) \end{bmatrix}$$

and θ is the angle ℓ makes with the x-axis (measured counterclockwise).

For example, what matrix represents a reflection in the line $y = x$? Here $\theta = 45°$.

$$\begin{bmatrix} \cos(2 \cdot 45°) & \sin(2 \cdot 45°) \\ \sin(2 \cdot 45°) & -\cos(2 \cdot 45°) \end{bmatrix} = \begin{bmatrix} \cos(90°) & \sin(90°) \\ \sin(90°) & -\cos(90°) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

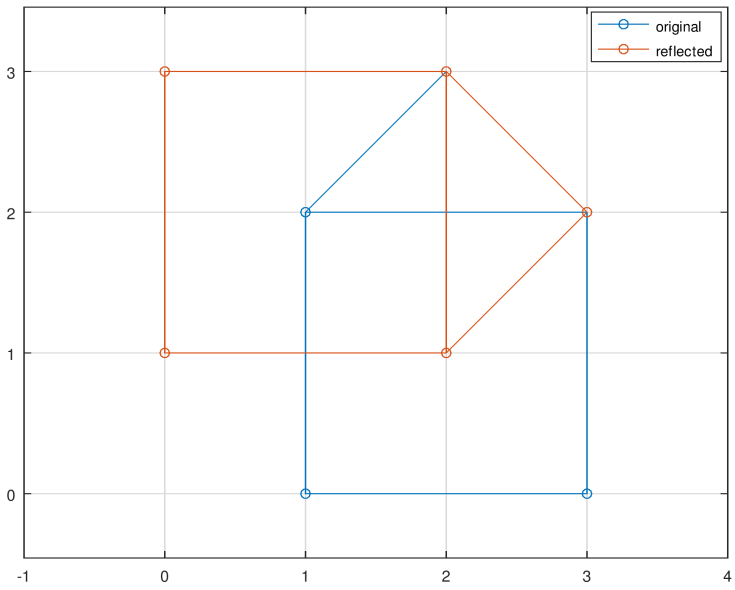


Figure 2.5: Reflection of the house graph

What is the effect of this matrix on a point (x, y)?

$$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} y \\ x \end{bmatrix}$$

We see that the point is indeed reflected in the line $y = x$.

Example 2.3.2. Reflect the house graph in the line $y = x$.

Solution. With the data matrix D and the original x and y vectors already defined, and using R as determined above, we have:

```
>> R = [0  1; 1  0]
R =

     0     1
     1     0

>> RD = R*D;
>> x1 = RD(1, :);
>> y1 = RD(2, :);
>> plot(x, y, 'o-', x1, y1, 'o-')
>> axis([-1 4 -1 4], 'equal');
>> grid on;
>> legend('original', 'reflected')
```

The result is shown in Figure 2.5. □

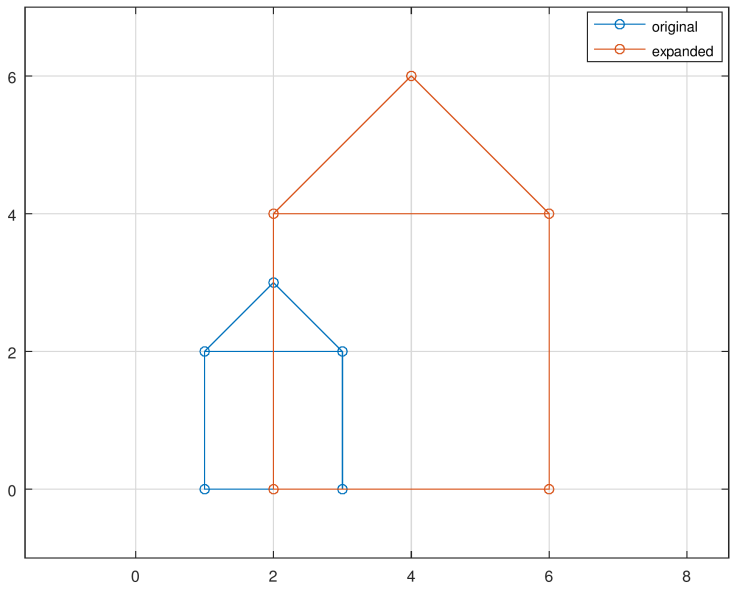


Figure 2.6: Dilation of the house graph

2.3.3 Dilation

Dilation (i.e., expansion or contraction) can also be accomplished by matrix multiplication. Let

$$T = \begin{bmatrix} k & 0 \\ 0 & k \end{bmatrix}$$

Then the matrix product TD will expand or contract D by a factor of k.

Example 2.3.3. Expand the house graph by a factor of 2.

Solution. To scale by a factor of 2, we only need to multiply D by the matrix $\begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}$.

```
>> T = [2  0; 0  2]
T =

     2     0
     0     2

>> TD = T*D;
>> x1 = TD(1, :); y1 = TD(2, :);
>> plot(x, y, 'o-', x1, y1, 'o-')
>> axis([-1 7 -1 7], 'equal');
>> grid on;
>> legend('original', 'expanded')
```

The result is shown in Figure 2.6. □

2.3.4 Linear and nonlinear transformations

Rotation, reflection, and dilation are examples of *linear transformations*, and therefore can be represented as matrix products. However, not all of the graphical operations we are interested in are linear in this sense. In particular, translation is a nonlinear transformation which *could* instead be accomplished by matrix addition. But, in fact even a translation can be completed using a special kind of matrix product with *homogeneous coordinates*. Refer to Exercises 13–14 for some of the details and an example. This method preserves some of the properties of the linear transformations of the preceding sections.

If we use only multiplication for transformations, the composition of several transformations can be handled with the relatively simple operation of composing matrix multiplications. Furthermore, inverse transformations are easily produced by inverting the original transformation matrix[2]. For example, if T is a translation, R is a rotation, and S is a stretch, the combined operations of first translating, then rotating, then stretching can be completed with the matrix SRT and a data matrix D can be transformed with the product $(SRT)D$. The inverse of these combined operations is $(SRT)^{-1} = T^{-1}R^{-1}S^{-1}$. Note that order matters!

[2]Rotation and reflection matrices, in particular, are easily inverted, since they are orthogonal (see Section 5.3.1).

Chapter 2 Exercises

1. Solve the system of equations using Gaussian elimination row operations

$$\begin{cases} -x_1 + x_2 - 2x_3 &= 1 \\ x_1 + x_2 + 2x_3 &= -1 \\ x_1 + 2x_2 + x_3 &= -2 \end{cases}$$

   To document your work in Octave, click "select all," then "copy" under the edit menu, and paste your work into a Word or text document. After you have the row echelon form, solve the system by hand on paper, using backward substitution.

2. Use the multipliers from Exercise 1 to write an LU decomposition for

$$A = \begin{bmatrix} -1 & 1 & -2 \\ 1 & 1 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

   Use this factorization to solve the system $A\mathbf{x} = \mathbf{b}$, where $\mathbf{b} = \langle -3, 1, 4 \rangle$.

3. Consider the system of linear equations $A\mathbf{x} = \mathbf{b}$, where

$$A = \begin{bmatrix} 1 & -3 & 5 \\ 2 & -4 & 3 \\ 0 & 1 & -1 \end{bmatrix} \text{ and } \mathbf{b} = \begin{bmatrix} 1 \\ -1 \\ 3 \end{bmatrix}$$

   Solve the system using left division. Then, construct an augmented matrix B and use rref to row-reduce it. Compare the results.

4. Use LU decomposition to solve the system from Exercise 3. Use Octave's [L U P] = lu(A) command. To use $PA = LU$ to solve $A\mathbf{x} = \mathbf{b}$, first multiply through by P, then replace PA with LU:

$$\begin{aligned} A\mathbf{x} &= \mathbf{b} \\ PA\mathbf{x} &= P\mathbf{b} \\ LU\mathbf{x} &= P\mathbf{b} \end{aligned}$$

   Solve $L\mathbf{y} = P\mathbf{b}$ using forward substitution. Then solve $U\mathbf{x} = \mathbf{y}$ using back substitution.

5. If A is a singular matrix, $A\mathbf{x} = \mathbf{b}$ has no solutions or infinitely many solutions depending on $\mathbf{b}$. How does Octave handle inconsistent systems, and in general, how does left division react to a singular coefficient matrix?

   (a) To explore this question, let's turn our previous system into an inconsistent one. Let A and $\mathbf{b}$ be the matrices from Exercise 3. To construct an inconsistent system, we will make one row of the coefficient matrix into a linear combination of some other rows, without making the corresponding adjustment to the right-side constants. Do the following:

```
>> A(1, :) = 3*A(2, :) - 4*A(3, :)
```

   Now $A\mathbf{x} = \mathbf{b}$ should be an inconsistent system. Try to solve it using left division. Does Octave provide a solution? Compare the results of left division with the row-reduced echelon form. How can you see that the system is inconsistent?

(b) Now let's consider a consistent system with infinitely many solutions. Keep A as above and make the corresponding adjustments to the right-side constants, yielding a system with infinitely many solutions.

```
>> b(1, :) = 3*b(2, :) - 4*b(3, :)
```

Now solve $A\mathbf{x} = \mathbf{b}$ using left division. Compare to the row-reduced echelon form.

The take away from these examples is that Octave will *always* give a solution when left division is used. If there are infinitely many solutions, a particular solution is given. If there are no solutions, Octave provides the least-squares (best-fitting) solution. It is always advisable to check the row-reduced echelon form of the coefficient matrix!

6. Octave can easily solve large problems that we would never consider working by hand. Let's try constructing and solving a larger system of equations. We can use the command rand(m, n) to generate an $m \times n$ matrix with entries uniformly distributed from the interval $(0, 1)$. If we want integer entries, we can multiply by 10 and use the floor function to chop off the decimal.

   Use this command to generate an augmented matrix M for a system of 25 equations in 25 unknowns:

   ```
   >> M = floor(10*rand(25, 26));
   ```

   Note the semicolon. This suppresses the output to the screen, since the matrix is now too large to display conveniently. Solve the system of equations using rref and/or left division and save the solution as a column vector $\mathbf{x}$.

7. Consider the following data.

x	2	3	5	8
y	3	4	4	5

   (a) Set up and solve the normal equations by hand to find the line of best fit, in $y = mx+b$ form, for the given data. Check your answer using polyfit(x, y, 1).

   (b) Compare to the solution found using Octave's left division operation directly on the relevant (inconsistent) system:

$$\begin{bmatrix} 2 & 1 \\ 3 & 1 \\ 5 & 1 \\ 8 & 1 \end{bmatrix} \cdot \begin{bmatrix} m \\ b \end{bmatrix} = \begin{bmatrix} 3 \\ 4 \\ 4 \\ 5 \end{bmatrix}$$

   (c) Plot a graph showing the data points and the regression line.

8. Use following commands to generate a randomized sample of 21 evenly spaced points from $x = 0$ to $x = 200$ with a high degree of linear correlation. We start with a line through the origin with random slope m, then add some "noise" to each y-value.

```
>> m = 2*rand - 1
>> x = [0 : 10 : 200]'
>> y = m*x + 10*rand(size(x))
```

Use the polyfit function to find the least squares regression line for this data. Plot the data and the best-fitting line on the same axes. Run the simulation several times, then save two example plots which exhibit greater and lesser amounts of variation from the line $y = mx$.

9. On July 4, 2006, during a launch of the space shuttle Discovery, NASA recorded the following altitude data[3].

Time (s)	Altitude (ft)
0	7
10	938
20	4,160
30	9,872
40	17,635
50	26,969
60	37,746
70	50,548
80	66,033
90	83,966
100	103,911
110	125,512
120	147,411

   (a) Find the quadratic polynomial that best fits this data. Use Octave to set-up and solve the normal equations. After you have the equations set up, solve using either the rref command or the left-division operator.

   (b) Plot the best-fitting parabola together with the given data points. Save or print the plot. Your plot should have labeled axes and include a legend.

   (c) Use the first and second derivatives of the quadratic altitude model from part (a) to determine models for the vertical component of the velocity and acceleration of the shuttle. Estimate the velocity two minutes into the flight.

10. There are many situations where the polynomial models we have considered so far are not appropriate. However, sometimes we can use a simple transformation to linearize the data. For example, if the points (x, y) lie on an exponential curve, then the points $(x, \ln y)$ should lie on a straight line. To see this, assume that $y = Ce^{kx}$ and take the logarithm of both sides of the equation:

$$\begin{aligned} y &= Ce^{kx} \\ \ln y &= \ln Ce^{kx} \\ &= \ln C + \ln e^{kx} \\ &= kx + \ln C \end{aligned}$$

Make the change of variables $Y = \ln y$ and $A = \ln C$. Then we have a linear function of the form

$$Y = kx + A$$

[3] https://www.nasa.gov/pdf/585034main_ALG_ED_SSA-Altitude.pdf

We can find the line that best fits the (x, Y)-data and then use inverse transformations to obtain the exponential model we need:

$$y = Ce^{kx}$$

where

$$C = e^A$$

Consider the following world population data[4]:

x = year	y = population (in millions)	$Y = \ln y$
1900	1650	7.4085
1910	1750	
1920	1860	
1930	2070	
1940	2300	
1950	2525	
1960	3018	
1970	3682	
1980	4440	
1990	5310	
2000	6127	
2010	6930	

(a) Fill in the blanks in the table with the values for $\ln y$. Note that in Octave, the log(x) command is used for the natural logarithm. Make a scatter plot of x vs. Y. This is called a semi-log plot. Is the trend approximately linear?

(b) Use the polyfit function to find the best-fitting line for the (x, Y)-data and add the graph of the line to your scatter plot from part (a). Save or print the plot. Your plot should have labeled axes and include a legend. Note that the vertical axis is the logarithm of the population. Give the plot the title "Semi-log plot."

(c) Use the data from part (b) to determine the exponential model $y = Ce^{kx}$. Plot the original data and the exponential function on the same set of axes. Save or print the plot. Your plot should have labeled axes and include a legend. Give the plot the title "Exponential plot."

(d) Use the model from part (c) to estimate the date when the global population reached 7 billion.

(e) Make a projection about when the global population will reach 10 billion.

11. Create a data matrix that corresponds to a picture of your own design, containing six or more edges. Plot it.

(a) Rotate the image through 45° and 180°. Plot the original image and the two rotations on the same axes. Include a legend.

(b) Expand your figure by a factor of 2, then reflect the expanded figure in the x-axis. Plot the original image, the expanded image, and the reflected expanded image on the same axes. Include a legend.

[4] https://esa.un.org/unpd/wpp/

12. Let $f(x) = x^2$, where $-3 \leq x \leq 3$. Use a rotation matrix to rotate the graph of the function through an angle of 90°. Plot the original and rotated graphs on the same axes. Include a legend.

13. Let the point (x, y) be represented by the column vector $\begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$. These are known as *homogeneous coordinates*. Then the translation matrix

$$T = \begin{bmatrix} 1 & 0 & h \\ 0 & 1 & k \\ 0 & 0 & 1 \end{bmatrix}$$

is used to move the point (x, y) to $(x + h, y + k)$ as follows:

$$\begin{bmatrix} 1 & 0 & h \\ 0 & 1 & k \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x + h \\ y + k \\ 1 \end{bmatrix}$$

Use a translation matrix and homogeneous coordinates to shift the graph you created in problem 11 as follows: shift 3 units left and 2 units up.

14. The translation method described in problem 13 can be combined with a rotation matrix to give rotations around an arbitrary point. Suppose for example that we wished to rotate the house graph from Figure 2.3 about the center of the rectangular portion (coordinates $(2, 1)$ in the original figure). This can be done by using homogeneous coordinates and a translation T to move the figure, then a rotation matrix R for the rotation. The form of R is now

$$R = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

The shifted and rotated figure is then given by $(RT)D$. To shift back to the original position, an inverse transformation T^{-1} is used. Thus the rotated image can be found by computing $(T^{-1}RT)D$. Use this method to rotate the house graph 90° about the point $(2, 1)$. Show the combined transformation matrix $T^{-1}RT$ and the results.

Chapter 3

Single variable calculus

3.1 Limits, sequences, and series

Octave is an excellent tool for many types of numerical experiments. Octave is a full-fledged programming language supporting many types of loops and conditional statements. However, since it is a vector-based language, some things that would be done using loops in Fortran or other traditional languages can be "vectorized." As an example, let's construct some numerical evidence to determine the value of the following limit:

$$\lim_{n\to\infty}\left(1+\frac{1}{n}\right)^n$$

We need to evaluate the expression for a series of larger and larger n-values. Here is what we mean by vectorized code: Instead of writing a loop to evaluate the function multiple times, we will generate a vector of input values, then evaluate the function using the vector input. This produces code that is easier to read and understand, and executes faster, due to Octave's underlying efficient algorithms for matrix operations.

First, we need to define the function. There are a number of ways to do this. The method we use here is known as an *anonymous function*. This is a good way to quickly define a simple function.

```
>> f = @(n) (1 + 1./n).^n;  % anonymous function
```

Note the use of elementwise operations[1]. We have named the function f. The input variable is designated by the @-sign followed by the variable in parentheses. The expression that follows gives the rule to be used when the function is evaluated. Now we can evaluate f at a single input value, or for an entire vector of input values.

Next, we create an index variable, consisting of the integers from 0 to 9.

[1]Elementwise operations are used throughout this chapter, since we are primarily evaluating functions elementwise at a numeric input vector, not doing matrix operations. Refer to Section 1.4.1.

```
>> k = [0 : 1 : 9]'  % index variable
k =

    0
    1
    2
    3
    4
    5
    6
    7
    8
    9
```

The syntax [0 : 1 : 9] produces a row vector that starts at 0 and increases by an increment of 1 up to 9. If not otherwise specified, the default step size is 1, so we could write simply [0 : 9]. Notice that we have used the transpose operation, only because our results will be a little easier to read as column vectors. Now, we'll take increasing powers of 10, which will be the input values, then evaluate $f(n)$.

```
>> format long % display additional decimal places
>> n = 10.^k   % sequence of increasing input values
n =

             1
            10
           100
          1000
         10000
        100000
       1000000
      10000000
     100000000
    1000000000

>> f(n)        % sequence of function values
ans =

   2.00000000000000
   2.59374246010000
   2.70481382942153
   2.71692393223552
   2.71814592682436
   2.71826823719753
   2.71828046915643
   2.71828169398037
   2.71828178639580
   2.71828203081451

>> format      % return to standard 5-digit display
```

This is good evidence that the limit converges to a finite value that is approximately 2.71828... You (hopefully!) recognize the number as e.

Similar methods can be used for numerical exploration of sequences and series, as we show in the following examples.

Example 3.1.1. Let $\sum_{n=2}^{\infty} a_n$ be the series whose nth term is $a_n = \dfrac{1}{n(n+2)}$. Find the first ten terms, the first ten partial sums, and plot the sequence and partial sums.

Solution. To do this, we will define an index vector n from 2 to 11, then calculate the terms.

```
>> n = [2 : 11]';        % index
>> a = 1./(n.*(n + 2)) % terms of the sequence
a =

    0.1250000
    0.0666667
    0.0416667
    0.0285714
    0.0208333
    0.0158730
    0.0125000
    0.0101010
    0.0083333
    0.0069930
```

If we want to know the 10th partial sum, we need only type sum(a). If we want to produce the sequence of partial sums, we need to make careful use of a loop. We will use a for loop with index i from 1 to 10. For each i, we produce a partial sum of the sequence a_n from the first term to the ith term. The output is a 10-element vector of these partial sums.

```
>> for i = 1:10
     s(i) = sum(a(1:i));
   end
>> s' % sequence of partial sums, displayed as a column vector
ans =

    0.12500
    0.19167
    0.23333
    0.26190
    0.28274
    0.29861
    0.31111
    0.32121
    0.32955
    0.33654
```

Finally, we will plot the terms and partial sums, for $2 \le n \le 11$.

```
>> plot(n, a, 'o', n, s, '+')
>> grid on
>> legend('terms', 'partial sums')
```

The result is shown in Figure 3.1. □

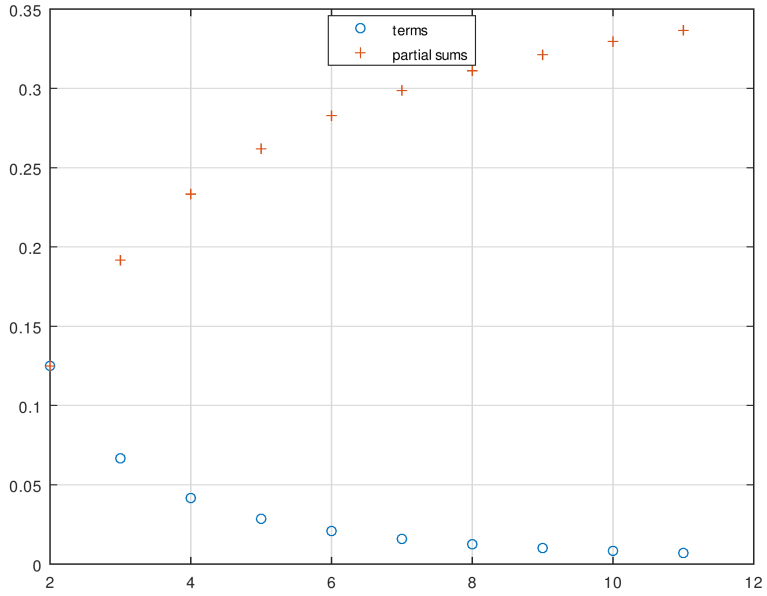


Figure 3.1: Plot of a sequence and its partial sums

An advantage of using a language like Octave is that it is simple to determine the sum of many terms of a series. If the series is known to converge, this can help give an estimate for the sum.

Example 3.1.2. Find the sum of the first 1000 terms of the harmonic series.

$$\sum_{n=1}^{1000} \frac{1}{n}$$

Solution. We only need to generate the terms as a vector, then take its sum. Recall that ending a command with a semicolon prevents the output from being displayed on screen, done here since our sequence is now much too long to display conveniently.

```
>> n = [1 : 1000];
>> a = 1./n;
>> sum(a)
ans =  7.4855
```

This is of course not an estimate for the sum of the infinite series, since, by the integral test, we know the series diverges. However, we can explore just how slowly this particular series diverges. The first 1000 terms sum to only about 7.5. Let's try adding more terms:

```
>> n = [1 : 1e6];  % using scientific notation for upper limit
>> a = 1./n;
>> sum(a)
ans =  14.393
```

The sum of the first million terms is still under 15! □

3.2 Numerical integration

3.2.1 Quadrature

Octave has several built-in functions to calculate definite integrals. We will use the built-in quad command. "Quad" is short for *quadrature*, which is a historical term referring to the process of calculating area by dividing into rectangles.

Example 3.2.1. Estimate $\int_0^{\pi/2} e^{x^2}\cos(x)\,dx$ using Octave's quad algorithm.

Solution. The correct syntax is quad('f', a, b). We need to first define the function.

```
>> function y = f(x)
     y = exp(x.^2).*cos(x);
   end
>> quad('f', 0, pi/2)
ans =  1.8757
```

Note that the function exp(x) is used for e^x. In this example, we used the function ... end construction to define f. This is a versatile format that allows for multiple operations and outputs. We could have instead used an anonymous function, but note that no quotes are used around the name f if using an anonymous function with quad. □

3.2.2 Octave scripts

Now suppose we want to write our own code for numeric integration. The midpoint rule, trapezoid rule, and Simpson's rule are common algorithms used for numerical integration. These types of algorithms are easily implemented in *Octave script* files.

Let $\{a = x_0, x_1, x_2, \ldots, x_n = b\}$ be a partition of $[a, b]$ into n subintervals, each of width $\Delta x = \dfrac{b-a}{n}$. Then $\int_a^b f(x)\,dx$ can be approximated as follows.

MIDPOINT RULE:

$$\Delta x\,[f(m_1) + f(m_2) + \cdots + f(m_n)]$$

where m_i is the midpoint of the ith subinterval.

TRAPEZOID RULE:

$$\frac{\Delta x}{2}\,[f(x_0) + 2f(x_1) + 2f(x_2) + \cdots + 2f(x_{n-1}) + f(x_n)]$$

SIMPSON'S RULE:

$$\frac{\Delta x}{3}\,[f(x_0) + 4f(x_1) + 2f(x_2) + 4f(x_3) + \cdots + 2f(x_{n-2}) + 4f(x_{n-1}) + f(x_n)]$$

where $x_i = a + i\Delta x$.

Octave script files are plain text files containing a series of Octave commands. A script file needs to have a ".m" extension (not the .txt used by default in Windows for text files) and cannot begin with the keyword function. You can use any text editor, such as Notepad, Notepad++, or Emacs, but the Octave GUI has its own built-in text editor which can be accessed by changing to the "Editor" tab option displayed below the main command window. The built-in editor is ideal for creating, editing, and running .m files and will automatically color code comments and key words.

Example 3.2.2. Write an Octave script to calculate a midpoint rule approximation of

$$\int_0^{\pi/2} e^{x^2} \cos(x)\, dx$$

using $n = 100$.

Solution. The basic strategy is to use a for loop that adds an additional function value to a running total with each iteration. Then the final answer is found by multiplying the sum by Δx.

The following code can be used. Switch to the editor tab and enter the code in a plain text file. Save it as `midpoint.m`. It must be placed in your working directory, then it can be run by typing midpoint at the command prompt, or by clicking the "save and run" button on the editor toolbar (be sure to switch back to the command window to see the output).

Octave Script 3.1: Midpoint rule approximation

```
% file 'midpoint.m'
% calculates a midpoint rule approximation of
%   the integral from 0 to pi/2 of f(x) = exp(x^2)cos(x)
%   --traditional looped code

% set limits of integration, number of terms and delta x
a = 0
b = pi/2
n = 100
dx = (b - a)/n

% define function to integrate
function y = f(x)
  y = exp(x.^2).*cos(x);
end

msum = 0;       % initialize sum
m1 = a + dx/2; % first midpoint

% loop to create sum of function values
for i = 1:n
  m = m1 + (i - 1)*dx; % calculate midpoint
  msum = msum + f(m);  % add to midpoint sum
end

% midpoint approximation to the integral
approx = msum*dx
```

Now run midpoint.m.

```
>> midpoint
a = 0
b =  1.5708
n =  100
dx =  0.015708
approx =  1.8758
```

□

The traditional code works fine, but because Octave is a vector-based language, it is also possible to write vectorized code that does not require any loops.

Example 3.2.3. Write a vectorized Octave script to calculate a midpoint rule approximation of

$$\int_0^{\pi/2} e^{x^2} \cos(x)\, dx$$

using $n = 100$.

Solution. Now our strategy is to create a vector of the x-coordinates of the midpoints. Then we evaluate f over this midpoint vector to obtain a vector of function values. The midpoint approximation is the sum of the components of the vector, multiplied by Δx.

Octave Script 3.2: Midpoint rule approximation - vectorized

```
% file 'midpoint2.m'
% calculates a midpoint rule approximation of
%   the integral from 0 to pi/2 of f(x) = exp(x^2)cos(x)
%   --vectorized code

% set limits of integration, number of terms and delta x
a = 0
b = pi/2
n = 100
dx = (b - a)/n

% define function to integrate
function y = f(x)
   y = exp(x.^2).*cos(x);
end

% create vector of midpoints
m = [a + dx/2 : dx : b - dx/2];

% create vector of function values at midpoints
M = f(m);

% midpoint approximation to the integral
approx = dx*sum(M)
```

This code will give the same results as the traditional looped code, but it executes faster, and is arguably more intuitive. □

3.3 Parametric, polar, and implicit functions

3.3.1 Parametric and polar plots

Curves defined by parametric and polar equations are usually studied in Calculus II. Such curves can be difficult to graph by hand! The plotting methods we used in Section 1.4 carry over easily to these new settings. For example, parametric equations for a *cycloid* are given by

$$\begin{aligned} x &= r(t - \sin(t)) \\ y &= r(1 - \cos(t)) \end{aligned}$$

Example 3.3.1. Graph three periods of a radius 2 cycloid.

Solution. The functions have period 2π, so we need $0 \le t \le 6\pi$ to see three full cycles. We need to define the parameter t as a vector over this range, then we calculate x and y, and plot x vs. y.

```
>> t = linspace(0, 6*pi, 50);
>> x = 2*(t - sin(t));
>> y = 2*(1 - cos(t));
>> plot(x, y)
>> axis('equal')
>> axis([0 12*pi 0 4])
```

The command axis('equal') is used to force an equal aspect ratio between the x- and y-axes. The result is shown in Figure 3.2. To see a simple animation of this plot, try comet(x, y). □

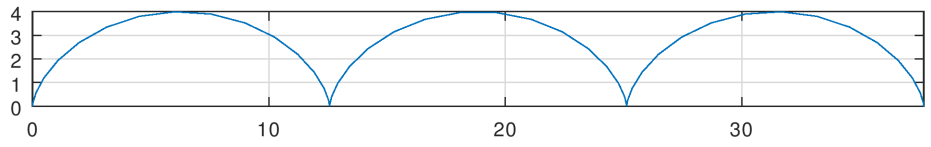


Figure 3.2: Graph of a cycloid

Polar graphs are handled in a similar way. For a function $r = f(\theta)$, we start by defining the independent variable θ, then we calculate r. To plot the graph, we calculate x and y using the standard polar identities $x = r\cos(\theta)$, $y = r\sin(\theta)$, then plot x vs. y.

Example 3.3.2. Plot the *limaçon* $r = 1 - 2\sin(\theta)$.

Solution. The needed commands are shown below and the graph is shown in Figure 3.3.

```
>> theta = linspace(0, 2*pi, 100);
>> r = 1 - 2*sin(theta);
>> x = r.*cos(theta);
>> y = r.*sin(theta);
>> plot(x, y)
```

Again, viewing an animation can be helpful. Just use the command comet(x, y). □

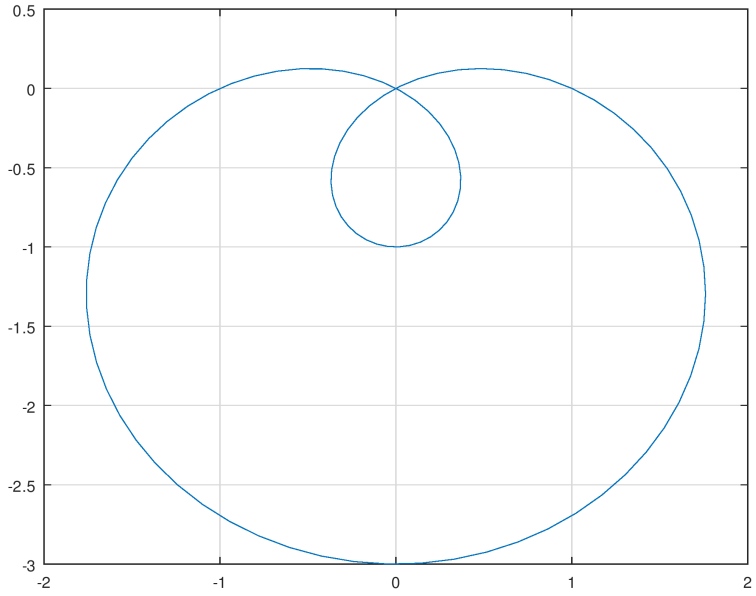


Figure 3.3: Graph of a limaçon on a rectangular grid

It is also possible to plot a function $r = f(\theta)$ on a set of polar axes using the polar command. For example, the following commands will produce a graph of the limaçon shown above on a polar grid. Try it! See Figure 3.4.

```
>> theta = linspace(0, 2*pi, 50);
>> r = 1 - 2*sin(theta);
>> polar(theta, r)
```

3.3.2 Implicit plots

Sometimes we need to plot a function defined implicitly by an equation of the form $f(x, y) = 0$. The easiest way to do this in Octave is with the ezplot command.

Example 3.3.3. Plot the curve defined by the equation

$$-x^2 - xy + x + y^2 - y = 1$$

Solution. To define the function as $f(x, y) = 0$, we subtract 1 from both sides of the equation.

```
>> f = @(x,y) -x.^2 - x.*y + x + y.^2 - y - 1
f =

@(x, y) -x .^ 2 - x .* y + x + y .^ 2 - y - 1

>> ezplot(f)
```

Run the ezplot command to see the results. Do you recognize the curve (see Figure 3.5)? □

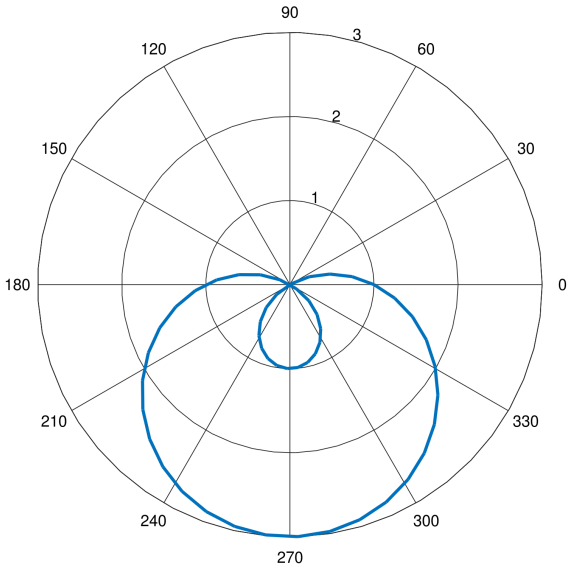


Figure 3.4: Graph of a limaçon on a polar grid

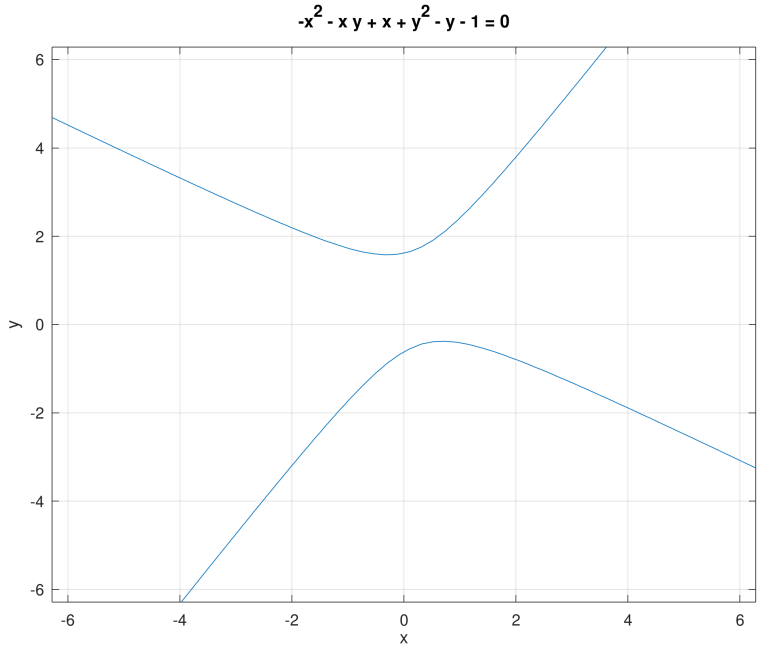


Figure 3.5: Implicit plot of a hyperbola

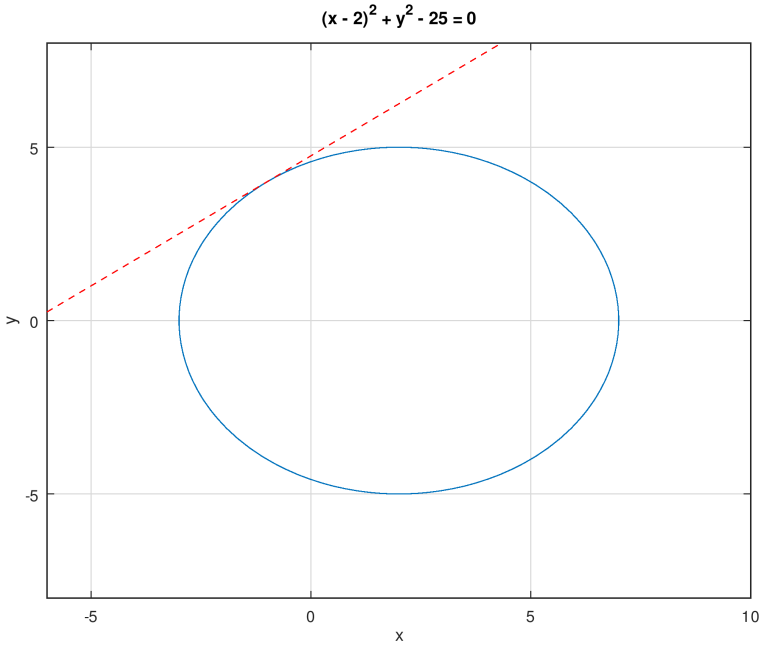


Figure 3.6: Implicit plot of a circle with tangent line

Example 3.3.4. Find the equation of the line tangent to the graph of the circle $(x-2)^2+y^2 = 25$, at the point $(-1, 4)$. Plot a graph of the circle and the tangent line on the same axes.

Solution. To plot the circle, we'll first define it as a function of the form $f(x, y) = 0$.

```
>> f = @(x, y) (x - 2).^2 + y.^2 - 25;
```

The center of the circle is at $(2, 0)$ and the radius is 5. We will set the axes of our plot to extend a few units beyond the circumference of the circle.

```
>> % implicit plot of f(x,y) = 0 over domain [-6, 10]x[-8, 8]
>> ezplot(f, [-6 10 -8 8])
```

Using implicit differentiation, the derivative is $y' = \dfrac{2-x}{y}$. At the point $(-1, 4)$, the slope is thus $3/4$. The equation of the tangent line is

$$y = \frac{3}{4}x + \frac{19}{4}$$

Now, add the tangent line to the graph.

```
>> x = [-6 : 10];
>> y = 3/4*x + 19/4;
>> hold on
>> plot(x, y, 'r--')
```

The result is shown in Figure 3.6. □

3.4 The symbolic package

While Octave is primarily numeric software, the Octave Forge *Symbolic* package allows Octave to function as a computer algebra system.

3.4.1 Installation

The package relies on the Python SymPy library, so installation can be a little tricky. But, if you already have a Python interpreter and SymPy, installation of the package in Octave only requires the command pkg install −forge symbolic.

If you don't want to do a separate installation of Python, a standalone installer for Windows users is available from the package developers that includes all dependencies. To install the package using the standalone Windows installer, download the file `symbolic-win-py-bundle-2.8.0.tar.gz` (or more recent version) from https://github.com/cbm755/octsympy/releases. Navigate to your download directory (the current working directory can be changed by clicking the folder icon at the top of the screen in the Octave GUI). Then in Octave, type:

```
>> pkg install symbolic-win-py-bundle-2.9.0.tar.gz
>> pkg load symbolic
```

To see if it is working, try declaring a symbolic variable. You should get a message indicating a Python communication link.

```
>> syms x
Symbolic pkg v2.8.0: Python communication link active, SymPy v1.3.
```

If you have trouble with the package installer, a "manual" installation is not too difficult, as follows:

1. Download and install Python (https://www.python.org/downloads/). Use one of the installer files appropriate for your system. Keep the default settings (includes PIP), and, if prompted, select the option to add Python to PATH.
2. At the system command prompt, navigate to the Python scripts folder (for example, something like `Programs\Python\Python37-32\Scripts`, depending on where it was installed) and type: `pip install symbolic`.
3. Then, in Octave, type: pkg install −forge symbolic. Test the installation by attempting to declare a symbolic variable as indicated above.

Once installed and operating, the package works much like MATLAB's Symbolic Toolbox.

3.4.2 Symbolic operations

A few basic examples showing the use of Octave as a computer algebra system are included below.

Example 3.4.1. Let $f(x) = x^3 + 3x^2 - 10x$.

(a) Evaluate $f(\frac{1}{2})$.

(b) Factor the expression and find all real zeros.

(c) Simplify the difference quotient and evaluate the limit $\lim_{h\to 0} \dfrac{f(x+h) - f(x)}{h}$.

Solution. The first step is to declare x as a symbolic variable with the command syms.

```
>> syms x                    % declare symbolic variable x
```

Now we define the expression. Notice that we do not need to worry about using elementwise operations here.

```
>> f = x^3 + 3*x^2 - 10*x  % define f as a symbolic expression
f = (sym)

   3       2
  x  + 3*x  - 10*x
```

The command to evaluate a symbolic expression is subs(f, x). In Octave, 1/2 evaluates to a decimal, hence a warning that floating-point values should not be passed to symbolic functions is raised if we enter subs(f, 1/2):

```
>> subs(f, 1/2)
warning: passing floating-point values to sym is dangerous, see 'help sym'
warning: called from
    double_to_sym_heuristic at line 50 column 7
    sym at line 379 column 13
    subs at line 178 column 9
```

To avoid this, we can enter 1/2 as a symbolic variable by using sym(1)/2, which keeps the fraction as an exact symbolic expression.

```
>> subs(f, sym(1)/2)
ans = (sym)  -33/8
```

Factoring is straightforward:

```
>> factor(f)
ans = (sym) x*(x - 2)*(x + 5)
```

To solve $f(x) = 0$, we use the solve command with the equality comparison, "==".

```
>> solve(f == 0, x)
ans = (sym 3x1 matrix)

[-5]
[ 0]
[ 2]
```

The solutions are easily seen to correspond to the factors in the factored form above.

We could create the difference quotient using the subs command described above. But, it is somewhat easier to instead redefine f as a *symbolic function*. Here is how:

```
>> syms f(x)                    % declare f as a symbolic function of x
>> f(x) = x^3 + 3*x^2 - 10*x    % define f(x)
f(x) = (symfun)

   3       2
  x  + 3*x  - 10*x
```

Now we can evaluate f at individual x-values, arrays of values, or at symbolic expressions. This makes it quite easy to build the difference quotient.

```
>> syms h
>> dq = (f(x + h) - f(x))/h
dq = (sym)

                3         2             3             2
  -10*h - x  - 3*x  + (h + x)  + 3*(h + x)
  ------------------------------------------
                        h
```

Now let's try to simplify that expression.

```
>> simplify(dq)
ans = (sym)

   2                          2
  h  + 3*h*x + 3*h + 3*x  + 6*x - 10
```

From here it is easy to see that $\lim_{h\to 0} \dfrac{f(x+h)-f(x)}{h} = 3x^2 + 6x - 10.$ □

In cases where a limit is less obvious, we may wish to try the symbolic limit function, but caution is required. In particular, as it is currently implemented[2], note that by default, only a right-hand limit is evaluated. For example, consider the following:

```
>> f(x) = x/abs(x);
>> limit(f, x, 0, 'left')
ans = (sym) -1

>> limit(f, x, 0, 'right')
ans = (sym) 1
```

The left and right limits do not agree. Clearly then $\lim_{x\to 0} f(x)$ does not exist. But, without qualifying the direction, limit(f, x, 0) gives the answer 1.

```
>> limit(f, x, 0)
ans = (sym) 1
```

[2]Symbolic pkg v2.9.0, as of this writing.

This behavior is counterintuitive, at best. A more sensible result would be NaN (not a number). When in doubt, check both left- and right-hand limits.

Next we will look at symbolic derivatives, as well as antiderivatives and definite integrals.

Example 3.4.2. Let $f(x) = x^2 \sin x$. Find each of the following:

(a) $f'(x)$

(b) $\int f(x)\, dx$

(c) $\int_0^{\pi/4} f(x)\, dx$

Solution. First we define the expression.

```
>> f = x^2*sin(x)
f = (sym)

     2
    x *sin(x)
```

Now, calculate the derivative using the diff command.

```
>> diff(f, x)
ans = (sym)

     2
    x *cos(x) + 2*x*sin(x)
```

Next, calculate the integrals using int.

```
>> int(f, x)
ans = (sym)

       2
  - x *cos(x) + 2*x*sin(x) + 2*cos(x)

>> int(f, x, 0, sym(pi)/4)
ans = (sym)

                 ---    2          ---
               \/ 2 *pi          \/ 2 *pi          ---
-2 - ------------------ + ---------------- + \/ 2
             32                   4
```

If you want to see the answer as a decimal approximation, type double(ans):

```
>> double(ans)
ans =   0.088755
```

This converts the result to a double-precision floating point value. □

Example 3.4.3. Show that

$$\int_{-r}^{r} 2\sqrt{r^2 - x^2}\, dx = \pi r^2$$

Solution. The integral, of course, represents the area of circle of radius r.

```
>> syms x r
>> f = 2*sqrt(r^2 - x^2)
f = (sym)

          ---------
         /  2     2
    2*\/   r  - x

>> int(f, -r, r)
```

If you run these commands as shown, you will get a rather complicated and unexpected answer. The problem is that we have implicitly assumed $r > 0$, but Octave does not know this. We can fix the problem by setting an assumption on r.

```
>> assume(r, 'positive')
>> int(f, -r, r)
ans = (sym)

        2
    pi*r
```

Now the result is as expected. Various other assumptions on symbolic variables are also possible (e.g., integer, nonzero, real, etc.). □

3.4.3 Plotting

The ezplot method we used in Section 3.3.2 for plotting implicit functions is also the easiest way to plot a symbolic function.

Example 3.4.4. Let $f(x) = x^3 + 3x^2 - 10x$. Graph f, f', and f'' on the same axes.

Solution. Once we've defined f, plotting the function is as simple as typing ezplot(f).

```
>> syms x
>> f = x^3 + 3*x^2 - 10*x
f = (sym)

   3       2
  x  + 3*x  - 10*x

>> ezplot(f)
```

The default plot is over $[-2\pi, 2\pi]$. If we want to see the function over a wider range, we can use ezplot(f, [a b]) to set the domain. In this case, the default domain covers the region of interest, but it will be helpful to adjust the viewpoint using the axis function. We would also like to change the line width. To do that, we will name the plot, then use that handle as a

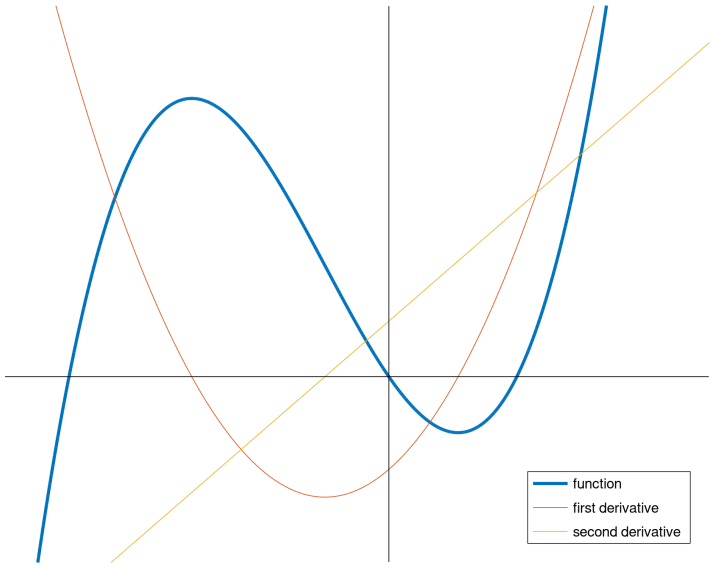


Figure 3.7: Graph of a polynomial and its derivatives

reference to the set function. Similar syntax can be used to adjust line width, color, and line style options.

```
>> h = ezplot(f);
>> set(h, 'linewidth', 2);
```

Now we just need to add the graphs of the first two derivatives.

```
>> hold on
>> ezplot(diff(f, x))
>> ezplot(diff(f, x, 2))
>> axis([-6 5 -20 40])
>> grid on
>> xlabel('x')
>> ylabel('y')
>> title('') % remove default ezplot title
>> legend('function', 'first derivative', 'second derivative')
>> legend('location', 'southeast')  % move legend to lower right
```

To more clearly show the relationship between these curves, it will be helpful to make the coordinate axes explicitly visible. We can use the method described in Chapter 1 Exercise 6.

```
>> plot([-6 5], [0 0], 'k', [0 0], [-20 40], 'k')
>> axis off
```

The graph is shown in Figure 3.7. □

3.4.4 Options

Notice that the output of symbolic operations in this section is displayed in a plain text approximation of standard mathematical notation. There are multiple display options available. The ASCII format is shown above. For a "prettier" format, you can change to a unicode pretty print format using the command sympref display unicode. But, this option does not work on all systems. For plain text without special formatting, use sympref display flat. Note that in this plain text mode, exponents are displayed as x**n instead of x^n, following the Python convention.

It is important to note that when the symbolic package is loaded, some commands shadow (override) a core library function of the same name. For instance, the normal operation of the command diff is taking the difference of adjacent elements in a vector (an operation useful for approximating derivatives numerically), but when the symbolic package is loaded, it becomes the symbolic differentiation operator. For help with symbolic functions, type help @sym/diff, help @sym/int, etc. For a complete list of available functions and options, refer to the documentation for the package[3].

[3] https://octave.sourceforge.io/symbolic/overview.html

Chapter 3 Exercises

1. Show (numerically) that $\lim_{\theta \to 0} \dfrac{\sin \theta}{\theta} = 1$.

2. Let $\sum a_n$ be the series whose nth term is $a_n = \dfrac{1}{2^n} - \dfrac{1}{3^n}, n \geq 1$. Find the first ten terms, the first ten partial sums, and plot the sequence and partial sums. Do you think the series converges? If so, what is the sum?

3. How many terms need to be included in the harmonic series to reach a partial sum that exceeds 10?

4. Write an Octave script based on a for-loop to calculate $\displaystyle\int_0^{\pi/2} e^{x^2} \cos(x)\, dx$ using the trapezoid rule with $n = 100$. Compare your answer to the midpoint approximation given in Examples 3.2.2 and 3.2.3. (Use the command format long to see more decimal places.)

5. Write a vectorized Octave script to calculate $\displaystyle\int_{-2}^{2} \frac{1}{\sqrt{2\pi}} e^{\frac{-x^2}{2}}\, dx$ using Simpson's rule with $n = 100$. Compare your answer to the midpoint approximation using Script 3.2. Which approximation seems to be most accurate, judged against Octave's quad algorithm?

6. Graph each equation.
   (a) $x = t^3, y = t^2$
   (b) $x = \sin(t), y = 1 - \cos(t)$
   (c) $r = \theta$
   (d) $r = \sin(2\theta)$
   (e) $r = \cos(7\theta/3)$
   (f) $x^2 = y^3 - 10y$

7. Octave scripts can be used for many problems in numerical analysis. Newton's method for root finding is a good example. Newton's method is an iterative process based on the formula
$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$
Starting from an initial guess of x_1, a sequence of approximations x_i is generated (refer to [1, §2.4] and [5, §4.1]).
   (a) The function $f(x) = x^3 + 5x^2 + x - 1$ has one positive root. Write an Octave script to find it using Newton's method.
   (b) Compare your answer to the result obtained with Octave's fsolve command.
```
>> fsolve('f', x1)   % solve f(x) = 0 numerically using initial
   guess x1
```
   (c) How many iterations of Newton's method were needed to obtain agreement with the fsolve result to five decimal places (using the same initial guess)?
   (d) Plot the function and its tangent lines at x_1, x_2, and x_3.

8. Let $f(x) = x^3 + 3x^2 - 10x$. Find the coordinates of any relative extrema and inflection points. Add markers and text labels to the graph in Figure 3.7 to highlight these points and their correspondence to the zeros of f' and f'', respectively.

9. The general *logistic growth equation* is

$$f(t) = \frac{C}{1 + Ae^{-kt}}$$

   (a) Let $A = 50$ and $k = 0.1$. Graph the logistic curves with $C = 100$, $C = 500$, and $C = 1000$ on a single set of axes. Include a legend. What does C represent?

   (b) Now, let $A = 50$ and $C = 100$. Graph the curves with $k = 0.1$, $k = 0.4$, and $k = 1$. How does the parameter k affect the shape of the curve?

   (c) Notice that in each case, the curve has a single inflection point. Find its coordinates, in terms of the parameters A, C, and k, using symbolic operations.

10. Recall that for $y = f(x)$, the arc length from $x = a$ to $x = b$ is given by

$$s = \int_a^b \sqrt{1 + \left(\frac{dy}{dx}\right)^2}\, dx$$

    (see [5, §7.4]).

    Let $f(x) = x^2$. Use symbolic functions to find an exact value for the arc length from $x = 0$ to $x = 1$.

11. The Taylor series for an infinitely differentiable function f at $x = a$ is given by

$$T(x) = \sum_{k=0}^{\infty} \frac{f^{(k)}(a)}{k!}(x - a)^k$$

    where $f^{(k)}$ is the kth derivative of f (see [5, §8.8]).

    Let $f(x) = e^{x^2}$.

    (a) Find (by hand) the first three nonzero terms of the Taylor series for f at $x = 0$.

    (b) Check your answer to part (a) by defining f as a symbolic function and using the taylor function:

```
>> syms x
>> f = exp(x^2)
>> T5 = taylor(f, x, 'order', 5)
```

    (c) Plot the function and the Taylor polynomial on the same axes.

    (d) Use the Taylor polynomial to approximate $\int_0^1 e^{x^2}\, dx$.

Chapter 4

Miscellaneous topics

4.1 Complex variables

Complex variables are ubiquitous in some engineering fields. Even if we attempt to limit our attention to real variables, some mathematical subjects cannot be fully understood without extending into the field of complex numbers. Examples include the eigenvalues and eigenvectors of matrices and the roots of polynomial equations.

The study of complex numbers begins with one basic definition:

$$i = \sqrt{-1}$$

The number i is called an *imaginary number*. A *complex number* $z = a + bi$ has real part a and imaginary part b. We will illustrate complex number arithmetic in Octave using some simple examples.

Example 4.1.1. Let $z_1 = 1 + 2i$ and $z_2 = 2 - 3i$. Find each of the following:

$$z_1 + z_2, z_2 - z_1, z_1 \cdot z_2, \text{and } z_1/z_2$$

Solution. Octave has no difficulty dealing with complex arithmetic. The variables i and j (not case-sensitive) are both by default recognized as the imaginary unit $\sqrt{-1}$.

First define the variables:

```
>> z1 = 1 + 2i;
>> z2 = 2 - 3i;
```

Now carry out the indicated operations:

```
>> z1 + z2
ans =  3 - 1i

>> z2 - z1
ans = 1 - 5i
```

```
>> z1*z2
ans =  8 + 1i

>> z1/z2
ans = -0.30769 + 0.53846i
```

Each result is expressed in the standard $a + bi$ format. The commands real(z) and imag(z) can be used to extract the real part a and complex part b, if needed. □

We can plot numbers in the complex plane using the compass command.

Example 4.1.2. Let $z_1 = 1+2i$ and $z_2 = 2-3i$. Plot z_1, z_2, and the sum z_1+z_2 in the complex plane.

Solution. We will show both variables and their sum on one set of axes.

```
>> z1 = 1 + 2i;
>> z2 = 2 - 3i;
>> compass(z1, 'b')
>> hold on
>> compass(z2, 'r')
>> compass(z1 + z2, 'k--')
>> legend('z_1', 'z_2', 'z_1+z_2')
```

The plot is shown in Figure 4.1. The horizontal axis is real and the vertical axis is imaginary.

It is interesting to note that the sum of the two variables in the complex plane is equivalent to the sum of two vectors in $\mathbf{R}^2$. Thus the compass command can also be used to illustrate vector arithmetic. The only difference is how we interpret the axes. □

Sometimes Octave will return complex results unexpectedly. For example, suppose we want to evaluate $\sqrt[3]{-8}$:

```
>> (-8)^(1/3)
ans =  1.0000 + 1.7321i
```

While we probably expected the answer -2, we can also easily verify that the cube of the given answer is indeed -8 (at least up to some minor round-off error[1]):

```
>> ans^3
ans = -8.0000e+00 + 2.2204e-15i
```

There are actually three cube roots of -8, and by default, Octave will return the one with the smallest argument (angle). See Exercise 2. If we simply want the real root, we can use the nthroot command.

```
>> % real cube root of -8
>> nthroot(-8, 3)
ans = -2
```

[1]Notice the format used for scientific notation: 2.2204e–15 = 2.2204×10^{-15}, effectively 0.

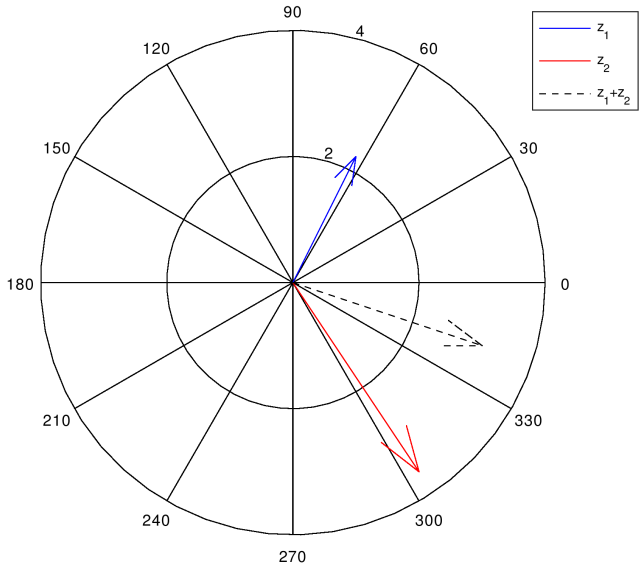


Figure 4.1: Addition in the complex plane

4.2 Special functions

Octave has many common *special functions* available, such as Bessel functions (bessel), the error function (erf), and the gamma function (gamma), to name a few.

For example, the *gamma function* is defined by

$$\Gamma(x) = \int_0^\infty t^{x-1} e^{-t}\, dt$$

This is an extension of the factorial function, since for positive integers n, the gamma function satisfies

$$\Gamma(n) = (n-1)!$$

Example 4.2.1. Graph $\Gamma(x+1)$ together on the same set of axes with the factorial function $x!$, for corresponding nonnegative integer values of x.

Solution. Both the gamma function and factorial function grow quite large very quickly, so we need to take care in selecting the domain. The gamma function is defined for positive and negative real numbers, while the factorial function is of course defined only for nonnegative integers. We will try the graph for $x \geq -5$ for the gamma function and $n = 0, 1, 2, 3, 4$ for the factorial.

Trial and error shows that a fine increment is needed for a smooth graph of the gamma function.

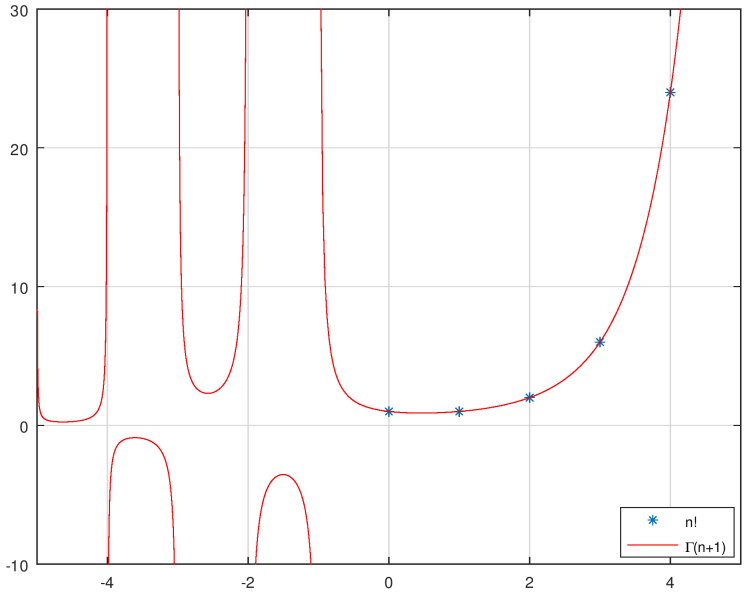


Figure 4.2: Improved graph of gamma function and factorial function

These are the basic commands needed:

```
>> n = [0 : 4];
>> x = linspace(-5, 5, 500);
>> plot(n, factorial(n), '*', x, gamma(x + 1))
>> axis([-5 5 -10 30]);
>> grid on;
>> legend('n!', 'gamma(n+1)', 'location', 'southeast')
```

Notice the vertical asymptotes at each negative integer. If you run the plot commands as shown above, you will see vertical line segments that are not a true part of the graph. If we don't want to see these, we can divide the domain into separate intervals with breaks at the discontinuities. This is somewhat tedious, but produces a more accurate graph, as shown in Figure 4.2.

Define the x-values as follows:

```
x1 = linspace(-5, -4, 200);
x2 = linspace(-4, -3, 200);
x3 = linspace(-3, -2, 200);
x4 = linspace(-2, -1, 200);
x5 = linspace(-1, 5, 200);
```

Then, plot x_1 vs. $\Gamma(x_1 + 1)$, x_2 vs. $\Gamma(x_2 + 1)$, etc., on the same set of axes. □

4.3 Statistics

4.3.1 Distribution of sample means

Octave has good capabilities for statistical analysis, and if you are proficient with Octave syntax, you will find it quite easy to solve many common statistical problems.

We'll start with something simple. Let's try rolling a six-sided die. The rand function returns a random value from the interval $(0, 1)$. To get integer values from 1 through 6, we multiply by 6 and add 1, then use the floor function to chop off the trailing decimal:

```
>> floor(6*rand + 1)
ans = 6  % the answer is random - your results will vary!
```

With the syntax rand(m, n), an $m \times n$ matrix of values is returned. Now let's try repeating the above experiment 100 times, storing the results in a column vector. We can analyze the results by looking at the sample mean, variance, and a histogram.

```
>> A = floor(6*rand(100, 1) + 1);
>> mean(A)
ans =  3.4100
>> var(A)
ans =  2.9514
>> hist(A, [1 2 3 4 5 6])
```

Your results will vary, but you should see something that looks close to a uniform distribution, such as in Figure 4.3. The vector [1 2 3 4 5 6] specifies the midpoints of the bins.

Now, let's use a loop to generate a distribution of 100 sample means.

```
>> for i = 1:100
       A = floor(6*rand(100, 1) + 1);
       d(i) = mean(A);
   end
>> hist(d)
```

Notice that the distribution of the sample means is approximately normal (Figure 4.4), even though the underlying distribution is not. We have just demonstrated the central limit theorem!

4.3.2 The standard normal distribution

We have seen that the rand function corresponds to a uniform distribution. Octave has many other discrete and continuous distributions available. For example, the function randn returns a matrix with normally distributed elements with mean 0 and standard deviation 1.

Example 4.3.1. Create a vector Z of 1000 elements from the standard normal distribution. Use the transformation $X = Z\sigma + \mu$ to generate a vector X of elements from a normal distribution with mean 400 and standard deviation 50. Compare the means and variances of X and Z. Plot histograms of Z and X.

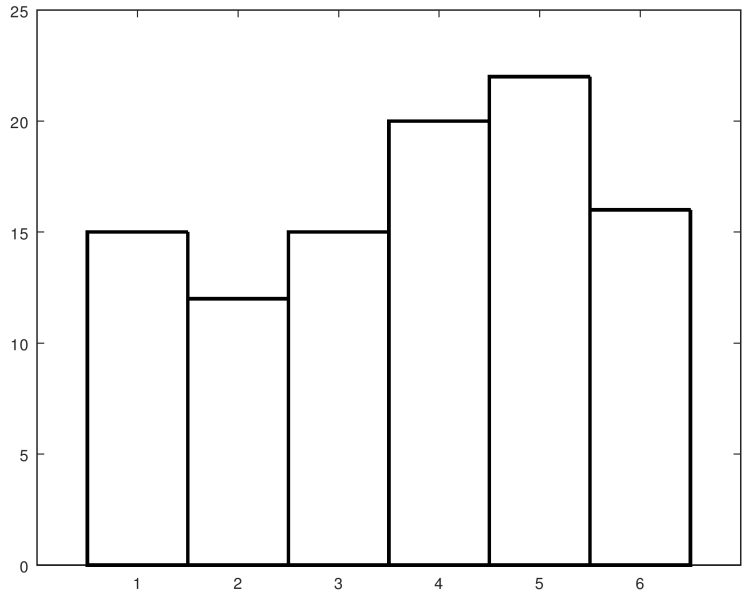


Figure 4.3: Results from 100 6-sided die trials

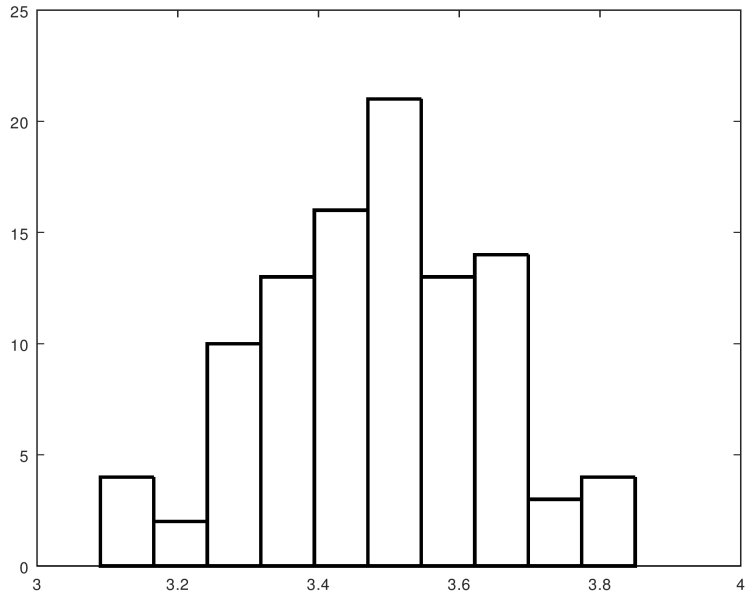


Figure 4.4: Distribution of sample means

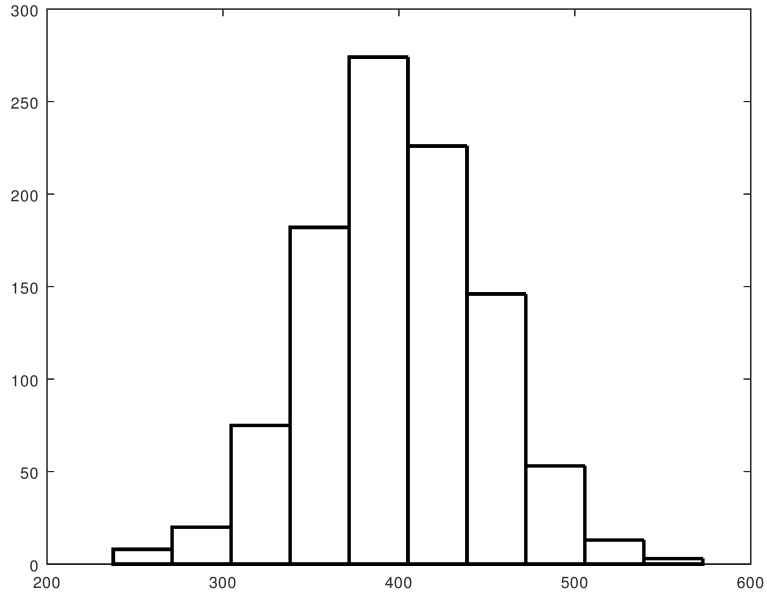


Figure 4.5: X-distribution

Solution. Here are the commands we need:

```
>> % sample 1000 elements from a standard normal distribution
>> Z = randn(1000, 1);

>> % transform the mean and standard deviation
>> mu = 400; sigma = 50;
>> X = Z*sigma + mu;

>> % review resulting sample mean and variance
>> format free;
>> mean([Z X])
ans =
   -0.00116119    399.942

>> var([Z X])
ans =
   1.04291    2607.28

>> % plot histograms
>> hist(Z)
>> hist(X)
```

The command format free changes from the default short form scientific notation. We can see Z has mean and variance near 0 and 1, respectively, while X has a mean near 400 and variance near 2500, as expected. The histograms are identical, except for the scale on the horizontal axis (see, for example, Figure 4.5). □

4.3.3 Linear regression

The polyfit command used in Section 2.2 (see Example 2.2.2) can be applied to linear regression problems. A linear function is a degree 1 polynomial, so we use the syntax polyfit(x, y, 1). To quantify the degree of linear correlation, we can calculate the correlation coefficient using corr(x, y).

Example 4.3.2. Let $x = \{5, 9, 18, 25, 32, 40, 53\}$ and $y = \{32, 28, 23, 20, 19, 18, 9\}$. Create a scatter plot of the data and calculate the correlation coefficient. Find the equation of the regression line and add it to the scatter plot.

Solution. First we enter the data and create the scatter plot.

```
>> x = [5 9 18 25 32 40 53];
>> y = [32 28 23 20 19 18 9];
>> plot(x, y, 'o');
```

Now, calculate the regression line and correlation coefficient.

```
>> P = polyfit(x, y, 1)
P =

   -0.42312   32.28685

>> r = corr(x, y)
r = -0.97394
```

The r-value suggests a strong negative linear correlation.

The equation of the regression line is $\hat{y} = -0.42312x + 32.28685$. We can add this to the plot using the polyval function to evaluate our regression equation at each x-value.

```
>> hold on;
>> y_hat = polyval(P, x);
>> plot(x, y_hat)
```

The scatter plot and regression line are shown in Figure 4.6. □

We can easily wrap these operations into our own user-defined regression function. Octave function files will be explained in more detail in Section 6.3. Enter the code in Script 4.1 in the text editor and save it as `linReg.m` in your current working directory. Test the function by running linReg(x, y), using x and y as given in Example 4.3.2.

Here are the results:

```
>> linReg(x, y);
y=ax+b
a=-0.423121
b=32.2869
r^2=0.948556
r=-0.973939
```

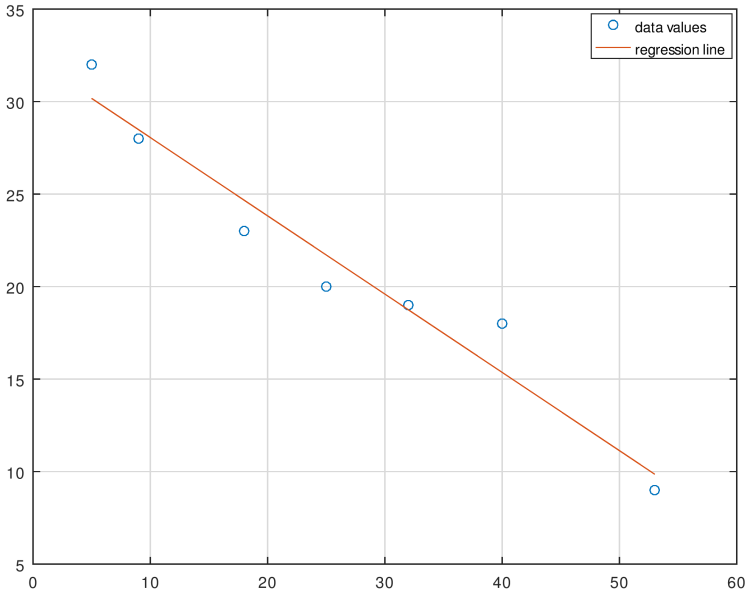


Figure 4.6: Scatter plot with regression line

Octave Script 4.1: Linear regression function

```
% function file 'linReg.m' runs a standard linear regression analysis
%   syntax: [P r] = linReg(x, y)
%   displays regression equation and correlation data, draws scatter plot
%   optional return values P: regression equation, r: correlation
    coefficient

function [P r] = linReg(x, y)
  % calculate regression and correlation
  P  = polyfit(x, y, 1);
  r = corr(x, y);

  % plot data and regression line
  figure()
  plot(x, y, 'o', x, polyval(P, x));
  legend('data values', 'regression line')
  grid on;

  % display results
  disp('y=ax+b')
  printf('%s%d\n', 'a=', P(1))
  printf('%s%d\n', 'b=', P(2))
  printf('%s%d\n', 'r^2=', r^2)
  printf('%s%d\n', 'r=', r)
end
```

Notice that Octave supports C-style formatted output strings using printf (and its relatives sprintf and fprintf). Refer to [3] for details.

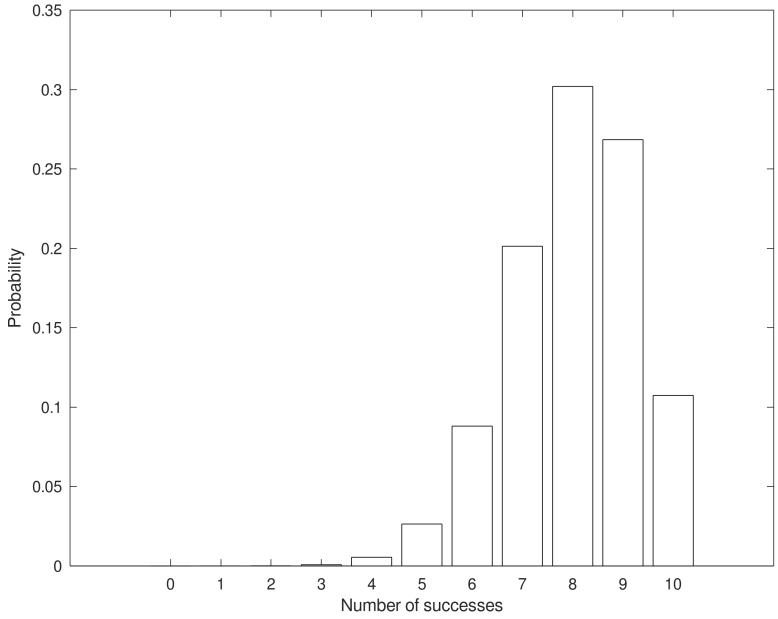


Figure 4.7: Binomial distribution with $n = 10$, $p = 0.8$

4.3.4 The binomial distribution

In a *binomial experiment*, we have a fixed number of independent trials. In each trial there are two possible outcomes, commonly designated success or failure. The probability of success on each trial is constant. Refer to [2] and [4] for details, as needed.

Example 4.3.3. Plot binomial distributions for $n = 10$, 25, and 50 trials with probability of success $p = 0.8$. What happens to the shape of the distribution as n increases?

Solution. The function binopdf(x, n, p) gives the probability of x successes in n trials of a binomial experiment with a probability of success p on each trial. Load the *statistics* package to access this function. The distributions can be plotted with the command bar(x, B), where x is the vector of possible outcomes and B is the corresponding vector of binomial probabilities.

First we generate a plot with $n = 10$.

```
>> pkg load statistics
>> n = 10; p = 0.8; x = [0 : n];
>> B = binopdf(x, n, p);
>> bar(x, B);
```

The graph is shown in Figure 4.7. Repeat the above steps for $n = 25$ and $n = 50$. You should see distributions whose shapes become progressively more normal. □

4.3.5 Hypothesis testing

Octave can handle many other statistical functions. As a final example, we will consider a simple hypothesis test. See [2] for background on the basic theory of statistical tests. To perform a t-test, we need to again ensure that the statistics package is loaded.

Example 4.3.4. Consider the following set of sample data, assumed to be from a normally distributed population:

$$\{24.9, 22.8, 16.2, 10.8, 32.0, 19.2\}$$

Test the following hypotheses at significance level $\alpha = 0.01$:

$$\begin{aligned} H_0: &\ \mu = 30 \\ H_a: &\ \mu < 30 \end{aligned}$$

Solution. Enter the data and calculate the mean.

```
>> x = [24.9 22.8 16.2 10.8 32.0 19.2]'
x =

   24.900
   22.800
   16.200
   10.800
   32.000
   19.200

>> mean(x)
ans =  20.983
```

Is $\overline{x} = 20.983$ good evidence that $\mu < 30$? We can use the ttest command. The basic format is ttest(X, mu), which will return 1 if the null hypothesis is rejected, 0 otherwise. The default options are for a two-tailed test using significance level $\alpha = 0.05$. Options are set using name-value string pairs. For this problem, we need to specify a left-tailed test and set a lower significance level. Here we ask for both the conclusion and the P-value, but additional output values are also possible (use help ttest for more details).

```
>> pkg load statistics
>> [h pval] = ttest(x, 30, 'tail', 'left', 'alpha', 0.01);
  h = 0
  pval =  0.014932
```

We can see that the P-value is greater than α and we fail to reject the null hypothesis. □

Chapter 4 Exercises

1. The polar form of a complex number is:

$$z = re^{i\theta}$$

where

$$re^{i\theta} = r\left(\cos(\theta) + i\sin(\theta)\right)$$

Octave can determine the magnitude (modulus) r and angle (argument) θ of a complex number z using the commands abs(z) and angle(z), respectively.

   (a) Write the polar form of $z_1 = 3 - 7i$ and $z_2 = 1 + 5i$.

   (b) Find $z_1 z_2$ in both polar and $a + bi$ form. How are the magnitudes and angles of each number related to the magnitude and angle of the product?

   (c) Find z_1/z_2 in both polar and $a + bi$ form. How are the magnitudes and angles of each number related to the magnitude and angle of the quotient?

2. A nonzero number (real or complex) x has n distinct nth roots. These are evenly spaced on a circle about the origin with radius equal to $\sqrt[n]{r}$, where r is the absolute value (or modulus) of x. Find the three complex cube roots of -8 and show them on a complex plane compass plot.

3. Graph the Bessel functions of the first kind $J_0(x)$, $J_1(x)$, and $J_2(x)$ on $[0, 20]$.

4. The gamma function can be used to calculate the "volume" (or "hypervolume") of an n-dimensional sphere. The volume formula is

$$V_n(a) = \frac{\pi^{n/2}}{\Gamma(\frac{n}{2} + 1)} \cdot a^n$$

where a is the radius, n is the dimension, and $\Gamma(n)$ is the gamma function.

   (a) Write a user-defined Octave function $V_n = f(n, a)$ that gives the volume of an n-dimensional sphere of radius a. Test it by computing the volumes of 2- and 3-dimensional spheres of radius 1. The answers should be π and $4\pi/3$, respectively.

   (b) Use the function to calculate the volume of a 4-dimensional sphere of radius 2 and a 12-dimensional sphere of radius 1/2.

   (c) For a fixed radius a, the "volume" is a function of the dimension n. For $n = 1, 2, \ldots, 20$, graph the volume functions for three different radii, $a = 1$, $a = 1.1$, and $a = 1.2$ (all on the same axes). Your graph should show points only for integer values of n and should have axis labels and a legend. Use the graph to determine the following limit:

$$\lim_{n\to\infty} V_n$$

   Does the answer surprise you?

5. Consider the following sample data: $\{46, 50, 66, 41, 47, 48, 48, 48, 48, 51, 48, 49, 47, 53, 50\}$. Plot a histogram using six bins. Find the mean and standard deviation.

6. Find the least-squares line $\hat{y} = ax + b$ that best fits the given set of points.

$$\{(-1, 5), (1, 4), (2, 2.5), (3, 0)\}$$

Include a plot of the data values and the least-squares line.

7. It is estimated that 7% of all patients using a particular drug will experience a mild side effect. A random sample of 12 patients using the drug is selected. Calculate the binomial distribution for $n = 12$ and $p = 0.07$. Plot a graph of the distribution. By summing various ranges of values from the distribution, determine each of the following:

   (a) The probability that no patients will have the mild side effect.

   (b) The probability that at most one patient will have the mild side effect.

   (c) The probability that no more than two patients will have the mild side effect.

   (d) The probability that at least three patients will have the mild side effect.

8. A manufacturer claims the life of a certain tire is greater than 50,000 miles. To test this claim, a sample of ten tires is tested. The following data are obtained from the sample:

Tread life (miles)
45,754
47,749
54,113
47,027
42,134
44,423
51,336
50,220
43,876
49,869

Test the manufacturer's claim using significance level $\alpha = 0.05$.

   (a) State the hypotheses you would use to test this claim.

   (b) Calculate the P-value and state your conclusion regarding the null hypothesis (i.e., reject or do not reject).

   (c) What do you conclude regarding the manufacturer's claim? State your answer in the context of the problem.

Chapter 5

Eigenvalue problems

5.1 Eigenvectors

We showed in Section 1.3.3 the use of eig(A) to find the eigenvalues of a square matrix A. You may have wondered about the corresponding eigenvectors. To find those, we use the eig command with two output arguments. Now the correct syntax is [v lambda] = eig(A). The first output will be a matrix whose columns represent the eigenvectors and the second output value will be a diagonal matrix with the eigenvalues on the diagonal.

```
>> A = [1  2  -3; 2  4  0; 1  1  1];
A =

    1    2   -3
    2    4    0
    1    1    1

>> [v lambda] = eig(A)    % 2-ouput form of eig command
v =

  -0.23995+0.00000i   -0.79195+0.00000i   -0.79195-0.00000i
  -0.91393+0.00000i    0.45225+0.12259i    0.45225-0.12259i
  -0.32733+0.00000i    0.23219+0.31519i    0.23219-0.31519i

lambda =

Diagonal Matrix

   4.52510+0.00000i                   0                   0
                  0   0.73745+0.88437i                   0
                  0                   0   0.73745-0.88437i
```

Perhaps we would like to see a matrix with real eigenvalues. We can construct a symmetric matrix (which must have real eigenvalues, as will be explained in Section 5.3.1) by multiplying a matrix and its transpose. For example:

```
>> C = A'*A
```

```
C =

    6    11    -2
   11    21    -5
   -2    -5    10

>> [v lambda] = eig(C)
v =

    0.876137    0.188733   -0.443581
   -0.477715    0.216620   -0.851390
   -0.064597    0.957839    0.279949

lambda =

Diagonal Matrix

    0.14970          0          0
          0    8.47515          0
          0          0   28.37516
```

Here again the diagonal entries of Λ are the eigenvalues and the corresponding columns of V are the associated eigenvectors. Each eigenvalue actually corresponds to an infinite family of eigenvectors, so Octave is only providing a representative vector, chosen according to criteria we will explore shortly. But first notice that they are normalized to unit length, and moreover, where possible, the collection is linearly independent.

5.2 Markov chains

Consider a sequence of random events, subject to the following conditions:

- A finite number of states are possible.
- At regular intervals an observation is made and the state of the system is recorded.
- For each state, we assign a probability of moving to each of the other states, or staying the same. The essential assumption is that these probabilities depend only on the current state.

Such a system is known as a *Markov chain*. Our problem is to predict the probability of future states of the system.

Suppose, for example, that we walk randomly along a four-block stretch of road in the following manner[1]. At intersections 2, 3, or 4 we either move to the left or to the right at random. Upon reaching the end of the road (intersections 1 or 5), we stop.

[1]The idea for this example comes from [4], which is an excellent open reference for more details about Markov chains and probability.

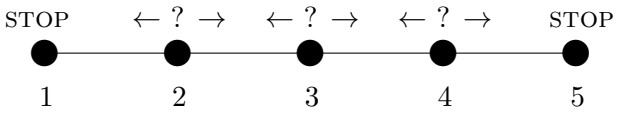


Figure 5.1: Random walk

Our goal is to predict where we will end up. We begin with a *probability vector*. For example, suppose we could start at any point with equal probability. Then the initial vector is $\langle 0.2, 0.2, 0.2, 0.2, 0.2\rangle$. On the other hand, we may know the initial state. Suppose we begin at intersection 3. Then the initial vector is $\langle 0, 0, 1, 0, 0\rangle$. In any case, we want to predict our location after k moves.

This is done by constructing a *transition matrix*. Form an $n \times n$ array whose ijth entry is the probability of moving from state i to j. Let T (for transition matrix) be the transpose of this matrix. The matrix product $T\mathbf{x}$ gives the new probability distribution after one time period. Continued left-multiplication by T gives the probabilities for future states. Thus, for any initial probability vector $\mathbf{x}$ and any positive integer k, the probability vector after k time periods is $\mathbf{y} = T^k\mathbf{x}$.

Example 5.2.1. For our random walk example, find the probability vector after five steps for each of these initial probability vectors:

$\mathbf{a} = \langle 0.2, 0.2, 0.2, 0.2, 0.2\rangle$

$\mathbf{b} = \langle 0.5, 0, 0, 0, 0.5\rangle$

$\mathbf{c} = \langle 0, 1, 0, 0, 0\rangle$

$\mathbf{d} = \langle 0, 0, 1, 0, 0\rangle$

Solution. We first form an array that records the probability of moving between positions.

		To				
		1	2	3	4	5
From	1	1	0	0	0	0
	2	0.5	0	0.5	0	0
	3	0	0.5	0	0.5	0
	4	0	0	0.5	0	0.5
	5	0	0	0	0	1

The transition matrix is the transpose.

$$T = \begin{bmatrix} 1 & 0.5 & 0 & 0 & 0 \\ 0 & 0 & 0.5 & 0 & 0 \\ 0 & 0.5 & 0 & 0.5 & 0 \\ 0 & 0 & 0.5 & 0 & 0 \\ 0 & 0 & 0 & 0.5 & 1 \end{bmatrix}$$

Notice that the sum of each column is 1. Now, the future state probabilities are easily computed as $T^k\mathbf{x}$, where $\mathbf{x}$ is the initial probability vector (expressed as a column).

```
>> T = [1 0.5 0 0 0; 0 0 0.5 0 0; 0 0.5 0 0.5 0; 0 0 0.5 0 0; 0 0 0
    0.5 1];
>> a = [0.2; 0.2; 0.2; 0.2; 0.2];
>> b = [0.5; 0; 0; 0; 0.5];
>> c = [0; 1; 0; 0; 0];
>> d = [0; 0; 1; 0; 0];
>> T^5*a
ans =

    0.450000
    0.025000
    0.050000
    0.025000
    0.450000

>> T^5*b
ans =

    0.50000
    0.00000
    0.00000
    0.00000
    0.50000

>> T^5*c
ans =

    0.68750
    0.00000
    0.12500
    0.00000
    0.18750

>> T^5*d
ans =

    0.37500
    0.12500
    0.00000
    0.12500
    0.37500
```

Notice that **b** is an *equilibrium vector* for which there is no change in future states. □

A probability vector $\mathbf{x}$ is an *equilibrium vector* if $\mathbf{x} = T\mathbf{x}$ where T is the transition matrix for the Markov chain. An equilibrium vector is one which results in no change moving to future states. Every Markov chain has at least one equilibrium vector and the eigenvalues of the transition matrix are the key to finding it.

Theorem 5.2.2. *Let T be the transition matrix for a Markov chain. Then $\lambda = 1$ is an eigenvalue of T. If $\mathbf{x}$ is an eigenvector for $\lambda = 1$ with nonnegative components that sum to 1, then $\mathbf{x}$ is an equilibrium vector for T.*

Example 5.2.3. Find an equilibrium vector for the Markov chain with transition matrix

$$T = \begin{bmatrix} 0.48 & 0.51 & 0.14 \\ 0.29 & 0.04 & 0.52 \\ 0.23 & 0.45 & 0.34 \end{bmatrix}$$

Solution.

```
>> T = [0.48 0.51 0.14; 0.29 0.04 0.52; 0.23 0.45 0.34]
T =

    0.480000    0.510000    0.140000
    0.290000    0.040000    0.520000
    0.230000    0.450000    0.340000

>> [v lambda] = eig(T)
v =

   -0.64840   -0.80111    0.43249
   -0.50463    0.26394   -0.81601
   -0.57002    0.53717    0.38351

lambda =

Diagonal Matrix

    1.00000          0          0
          0    0.21810          0
          0          0   -0.35810

>> x = v(:, 1)/sum(v(:, 1))
x =

    0.37631
    0.29287
    0.33082
```

Thus $\mathbf{x} = \langle 0.37631, 0.29287, 0.33082 \rangle$ is an equilibrium vector. Let's test it.

```
>> T^10*x
ans =

    0.37631
    0.29287
    0.33082

>> T^50*x
ans =

    0.37631
    0.29287
    0.33082
```

There is no change evident, so it seems to work! □

5.3 Diagonalization

Diagonal matrices have important properties. Some matrices can be transformed into a special diagonal matrix that shares some properties with the original matrix, in particular, its eigenvalues. The diagonalization problem is to find a matrix S such that

$$S^{-1}AS = \Lambda$$

where Λ is a diagonal matrix.

Theorem 5.3.1. *Let A be an $n \times n$ matrix with n linearly independent eigenvectors. Form an $n \times n$ matrix S whose columns are the eigenvectors of A. Then S is invertible and $S^{-1}AS = \Lambda$, where*

$$\Lambda = \begin{bmatrix} \lambda_1 & & & \\ & \lambda_2 & & \\ & & \ddots & \\ & & & \lambda_n \end{bmatrix}$$

and λ_i is the eigenvalue associated with the ith column of S. It follows that A can be written as $A = S\Lambda S^{-1}$.

Theorem 5.3.1 tells us how to diagonalize a square matrix. Notice that this can be done only for matrices that have enough independent eigenvectors. We need one more result.

Theorem 5.3.2. *If A is an $n \times n$ diagonalizable matrix and $A = S\Lambda S^{-1}$ and k is a positive integer, then*

$$A^k = S\Lambda^k S^{-1} = S \begin{bmatrix} \lambda_1^k & & & \\ & \lambda_2^k & & \\ & & \ddots & \\ & & & \lambda_n^k \end{bmatrix} S^{-1}$$

Theorem 5.3.2 shows how the diagonalized form can be used to simplify a particular computational problem, namely raising a matrix to a high power.

Example 5.3.3. Let $A = \begin{bmatrix} 7 & 8 \\ -4 & -5 \end{bmatrix}$. Find A^{100}.

Solution. Octave can solve such a problem easily.

```
>> A = [7  8; -4 -5]
A =

    7    8
   -4   -5

>> A^100
ans =

   1.0308e+048   1.0308e+048
  -5.1538e+047  -5.1538e+047
```

But how does Octave do this? Not by brute force, but by using Theorem 5.3.2. Here's how. First we need to calculate the eigenvalues and associated eigenvectors. Verify that the eigenvalues and eigenvectors are

$$\lambda_1 = 3, \mathbf{v}_1 = \begin{bmatrix} -2 \\ 1 \end{bmatrix}$$

$$\lambda_2 = -1, \mathbf{v}_2 = \begin{bmatrix} -1 \\ 1 \end{bmatrix}$$

Then $\Lambda = \begin{bmatrix} 3 & 0 \\ 0 & -1 \end{bmatrix}$. We form the matrix S using the eigenvectors:

$$S = \begin{bmatrix} -2 & -1 \\ 1 & 1 \end{bmatrix}$$

Now we need to calculate the inverse matrix. It is

$$S^{-1} = \begin{bmatrix} -1 & -1 \\ 1 & 2 \end{bmatrix}$$

Therefore the diagonalized form is

$$\begin{aligned} A &= S\Lambda S^{-1} \\ &= \begin{bmatrix} -2 & -1 \\ 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} 3 & 0 \\ 0 & -1 \end{bmatrix} \cdot \begin{bmatrix} -1 & -1 \\ 1 & 2 \end{bmatrix} \end{aligned}$$

So,

$$\begin{aligned} A^{100} &= S\Lambda^{100} S^{-1} \\ &= \begin{bmatrix} -2 & -1 \\ 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} 3 & 0 \\ 0 & -1 \end{bmatrix}^{100} \cdot \begin{bmatrix} -1 & -1 \\ 1 & 2 \end{bmatrix} \\ &= \begin{bmatrix} -2 & -1 \\ 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} 3^{100} & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} -1 & -1 \\ 1 & 2 \end{bmatrix} \\ &= \begin{bmatrix} -2 \cdot 3^{100} & -1 \\ 3^{100} & 1 \end{bmatrix} \cdot \begin{bmatrix} -1 & -1 \\ 1 & 2 \end{bmatrix} \\ &= \begin{bmatrix} 2 \cdot 3^{100} - 1 & 2 \cdot 3^{100} - 2 \\ -3^{100} + 1 & -3^{100} + 2 \end{bmatrix} \end{aligned}$$

Compare to the earlier Octave result:

```
>> [2*3^100-1  2*3^100-2; -3^100+1  -3^100+2]
ans =

   1.0308e+048   1.0308e+048
  -5.1538e+047  -5.1538e+047
```

This example shows some of the computational power of diagonalization. □

5.3.1 Orthogonal diagonalization

We have already observed that not all square matrices can be diagonalized. However, a certain class of square matrices always has a diagonalization, and this diagonalization has special properties. First, we need to recall a few definitions.

- A *symmetric matrix* is a square matrix A such that $A^T = A$. Recall that a matrix with real entries may have complex eigenvalues. That cannot happen with symmetric matrices. A real symmetric matrix has all real eigenvalues.
- An *orthogonal matrix* is a square matrix whose columns are *orthonormal* (orthogonal and length 1). An important property of orthogonal matrices is that their inverse is equal to their transpose: If A is orthogonal, then $A^{-1} = A^T$.

All symmetric matrices are diagonalizable. Moreover, we can say the following:

Theorem 5.3.4. *Let A be a symmetric matrix. Then A can be diagonalized as*

$$A = Q\Lambda Q^T$$

where Q is an orthogonal matrix whose columns are eigenvectors of A and Λ is a diagonal matrix with the associated eigenvalues on the diagonal.

Example 5.3.5. Find an orthogonal diagonalization for $A = \begin{bmatrix} 2 & -1 \\ -1 & 2 \end{bmatrix}$.

Solution. A has eigenvalues 3 and 1. The eigenvectors are $\begin{bmatrix} 1 \\ -1 \end{bmatrix}$ and $\begin{bmatrix} 1 \\ 1 \end{bmatrix}$. Notice that these are orthogonal. They are normalized by dividing by their length (both have length $\sqrt{2}$). Then A can be diagonalized as

$$\begin{aligned} A &= Q\Lambda Q^T \\ &= \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{-1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix} \cdot \begin{bmatrix} 3 & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{-1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix} \end{aligned}$$

The eigenvectors in this example were orthogonal since the eigenvalues were distinct. If the matrix A is symmetric, but has repeated eigenvalues, then the problem is a bit more difficult and finding a set of orthonormal eigenvectors requires the Gram-Schmidt process (see Section 5.5.1). We won't show the details here, but note that even in those cases, an orthonormal set of eigenvectors can still be found. □

Now, with these ideas in mind, let's take another look at the output of Octave's eig command.

```
>> A = [2  -1; -1  2]
A =

   2  -1
```

```
   -1    2

>> [v lambda] = eig(A)
v =

   -0.70711   -0.70711
   -0.70711    0.70711

lambda =

Diagonal Matrix

    1    0
    0    3
```

While the matrices are arranged slightly differently (the diagonalization is not unique), you should see that results given by Octave are precisely what is needed for the orthogonal diagonalization problem.

Example 5.3.6. Use Octave to orthogonally diagonalize $A = \begin{bmatrix} 3 & 3 \\ 3 & -1 \end{bmatrix}$.

Solution. If an orthogonal diagonalization is possible, Octave will return the output of the eig(A) command in that format. This explains why Octave chooses normalized vectors that form an orthogonal set, when possible.

```
>> A = [3  3; 3  -1]
A =

    3    3
    3   -1

>> [Q L] = eig(A)
Q =

    0.47186   -0.88167
   -0.88167   -0.47186

L =

Diagonal Matrix

   -2.6056         0
         0    4.6056

>> Q*L*Q'    % check the factorization by multiplying
ans =

    3.00000    3.00000
    3.00000   -1.00000
```

The reader can verify that Q is indeed orthogonal as required. □

5.4 Singular value decomposition

We are now prepared to tackle the singular value decomposition (SVD). This factorization is something of a generalized version of what we just did for symmetric matrices in Section 5.3.1. But, the singular value decomposition exists for any matrix; the matrix need not even be square. The key is to consider the matrices A^TA and AA^T. These are always square symmetric matrices, and so, can be orthogonally diagonalized.

There are many applications associated with the SVD. For example, Netflix recently sponsored a competition with a one million dollar prize to improve their movie recommendation algorithm. The team that won used a method based in part on the SVD[2], which can be used to discover hidden relationships among variables. We will consider applications to least-squares problems (Section 5.4.1) and image compression (Section 7.1).

Theorem 5.4.1. *Let A be an $m \times n$ matrix. The square roots of the nonzero eigenvalues of A^TA and AA^T (they are the same) are called the* singular values *of A,denoted $\sigma_1, \sigma_2, \ldots, \sigma_r$. Then A can be factored as*

$$A = U\Sigma V^T$$

where the columns of U are eigenvectors of AA^T, the columns of V are eigenvectors of A^TA, and the r singular values of A are on the diagonal of Σ. This factorization is called the singular value decomposition *of A.*

- U is $m \times m$ and orthogonal
- V is $n \times n$ and orthogonal
- Σ is $m \times n$ and diagonal of the special form

$$\Sigma = \begin{bmatrix} \sigma_1 & & & & \vdots \\ & \sigma_2 & & & 0 \\ & & \ddots & & \vdots \\ & & & \sigma_r & \\ & \cdots & 0 & \cdots & 0 \end{bmatrix}$$

If all the eigenvalues of A^TA are distinct, then the associated eigenvectors are "automatically" orthogonal. We only need to make them unit vectors. If there are repeated eigenvalues, it is still possible to choose orthogonal eigenvectors, but more advanced methods are needed (see Section 5.5.1). Our procedure starts with eigenvectors of A^TA, then appropriate orthogonal unit eigenvectors for AA^T are calculated using a simple formula. The number of singular values (nonzero eigenvalues) corresponds to the rank of the original matrix A. We will only consider examples where the number of singular values r is equal to m, the number of rows of A, otherwise, again, more advanced methods are required. We will consider a simple example using a 2×2

[2] http://buzzard.ups.edu/courses/2014spring/420projects/math420-UPS-spring-2014-gower-netflix-SVD.pdf

matrix, then see how Octave commands can be used to find the SVD for larger or more difficult matrices.

Here is the simplified procedure we will use:

1. Find A^TA.
2. Find the eigenvalues of A^TA. The square roots of these are the singular values σ_1, σ_2, ..., σ_r, arranged in decreasing order.
3. Find the corresponding eigenvectors and make them unit vectors $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n$.
4. Find the vectors $\mathbf{u}_i$ by computing $\mathbf{u}_i = \dfrac{1}{\sigma_i} A\mathbf{v}_i$.
5. Then $A = U\Sigma V^T$, where $\sigma_1, \sigma_2, \ldots, \sigma_r$ are on the diagonal of Σ and

$$U = \left[\begin{array}{c|c|c|c} \mathbf{u}_1 & \mathbf{u}_2 & \cdots & \mathbf{u}_m \end{array}\right]$$

$$V = \left[\begin{array}{c|c|c|c} \mathbf{v}_1 & \mathbf{v}_2 & \cdots & \mathbf{v}_n \end{array}\right]$$

- Remember to transpose V when you write the factorization.
- Remember to keep the eigenvalues and eigenvectors in their correct order.
- This simplified procedure only works if A^TA has no repeated eigenvalues and $r = m$.

Example 5.4.2. Let $A = \begin{bmatrix} 4 & 4 \\ -3 & 3 \end{bmatrix}$. Find the SVD via the simplified procedure outlined above, then compare to the results obtained using the Octave function svd.

Solution. We can readily verify that $\text{rank}(A) = 2$, so the matrix should have two singular values.

```
>> A = [4  4; -3  3]
A =

    4    4
   -3    3

>> ATA = A'*A
ATA =

   25    7
    7   25

>> [v lambda] = eig(ATA)
v =

   -0.70711    0.70711
    0.70711    0.70711

lambda =
```

```
Diagonal Matrix

    18     0
     0    32
```

Notice that the given eigenvectors are orthogonal unit vectors. However, the eigenvalues are not in decreasing order. So, we need to switch the order of both eigenvectors and singular values (they must be in decreasing order) as we build V and Σ.

```
>> Sigma = zeros(2, 2);
>> Sigma(1, 1) = sqrt(lambda(2, 2))
Sigma =

    5.65685    0.00000
    0.00000    0.00000

>> Sigma(2, 2) = sqrt(lambda(1, 1))
Sigma =

    5.65685    0.00000
    0.00000    4.24264

>> V(:, 1) = v(:, 2)
V =

    0.70711
    0.70711

>> V(:, 2) = v(:, 1)
V =

    0.70711   -0.70711
    0.70711    0.70711
```

Now we build U to complete the factorization.

```
>> U(:, 1) = 1/Sigma(1, 1)*A*V(:, 1)
U =

    1.00000
    0.00000

>> U(:, 2) = 1/Sigma(2, 2)*A*V(:, 2)
U =

    1.00000    0.00000
    0.00000    1.00000
```

Now, let's verify that $U\Sigma V^T = A$.

```
>> U*Sigma*V'
ans =
```

```
    4.0000    4.0000
   -3.0000    3.0000
```

Now that we have a rough sense of how an SVD is determined, let's try the built-in Octave function. The command [U S V] = svd(A) computes the SVD of a matrix A and stores the result in matrices U, S, and V. Let's use this command to find the SVD of the matrix A and verify that $USV^T = A$.

```
>> [U S V] = svd(A)    % 3-output format of svd command
U =

   -1    0
    0    1

S =

Diagonal Matrix

    5.6569         0
         0    4.2426

V =

   -0.70711   -0.70711
   -0.70711    0.70711

>> U*S*V'
ans =

    4.0000    4.0000
   -3.0000    3.0000
```

Notice that the factorization returned by svd is slightly different than we obtained above. This is normal: The SVD is not unique, due to variations in how representative eigenvectors are chosen. □

5.4.1 The pseudoinverse

In Section 2.2, we used the normal equations, $A^TA\mathbf{x} = A^T\mathbf{b}$, to solve least-squares problems. A potential problem with this approach is that the normal equations are typically *ill-conditioned*. This means that a small change in the data can lead to a large change in the numeric solution. This is bad! One way to avoid this computational problem is to use a generalized inverse known as the *pseudoinverse*, based on the SVD.

Theorem 5.4.3. *For an $m \times n$ matrix A with singular value decomposition $A = U\Sigma V^T$, the least-squares solution to the system $A\mathbf{x} = \mathbf{b}$ is given by*

$$\overline{\mathbf{x}} = A^+\mathbf{b}$$

where

$$A^+ = V\Sigma^+U^T$$

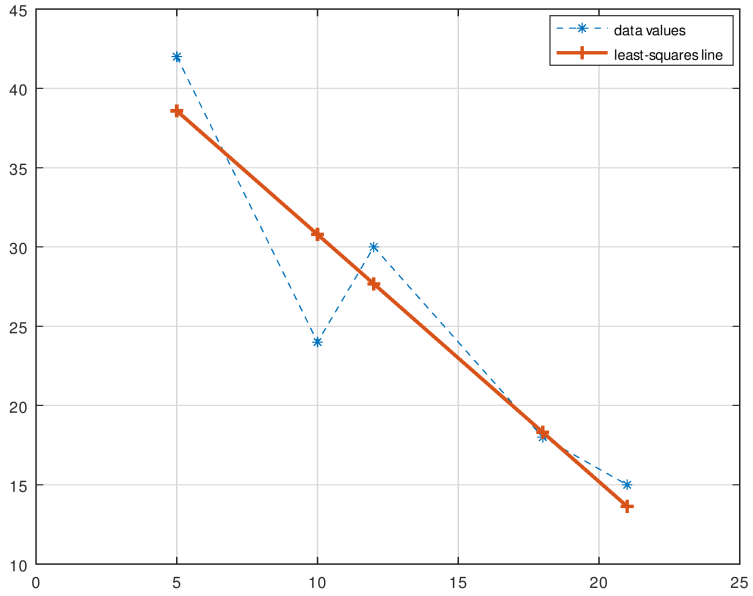


Figure 5.2: Regression line and original data

and Σ^+ *is the* $n \times m$ *matrix found by transposing* Σ *and taking the reciprocals of the singular values:*

$$\Sigma^+ = \begin{bmatrix} 1/\sigma_1 & & & & \vdots \\ & 1/\sigma_2 & & & 0 \\ & & \ddots & & \vdots \\ & & & 1/\sigma_r & \\ & \cdots & 0 & \cdots & 0 \end{bmatrix}$$

The matrix A^+ *is called the* pseudoinverse *or* Moore-Penrose inverse *of* A.

Example 5.4.4. Consider the following sample data.

x	5	10	12	18	21
y	42	24	30	18	15

Find a linear equation of the form $y = ax + b$ to model this data.

Solution. The given points yield a system $A\mathbf{p} = \mathbf{y}$, with

$$A = \begin{bmatrix} 5 & 1 \\ 10 & 1 \\ 12 & 1 \\ 18 & 1 \\ 21 & 1 \end{bmatrix}, \mathbf{p} = \begin{bmatrix} a \\ b \end{bmatrix}, \text{ and } \mathbf{y} = \begin{bmatrix} 42 \\ 24 \\ 30 \\ 18 \\ 15 \end{bmatrix}$$

Enter $\mathbf{x}$, $\mathbf{y}$, and A in Octave. We need to find the SVD of A:

```
>> [U S V] = svd(A)
U =

   -0.156839   -0.767088   -0.379243   -0.354748   -0.342501
   -0.311700   -0.407114   -0.118019    0.444130    0.725204
   -0.373645   -0.263125    0.877307   -0.107405   -0.099761
   -0.559478    0.168843   -0.176557    0.585728   -0.533129
   -0.652394    0.384827   -0.203488   -0.567705    0.250187

S =

Diagonal Matrix

   32.22136         0
          0   0.88546
          0         0
          0         0
          0         0

V =

   -0.997966    0.063748
   -0.063748   -0.997966
```

Next, we construct Σ^+ by taking the transpose and reciprocal of Σ:

```
>> Sp = (1./S)';          % note: division by 0 returns inf
>> Sp(isinf(Sp)) = 0      % set all the instances of inf to 0
Sp =

   0.03104   0.00000   0.00000   0.00000   0.00000
   0.00000   1.12936   0.00000   0.00000   0.00000
```

Note the "trick" used to handle the reciprocal operation. Now calculate the pseudoinverse:

```
>> Ap = V*Sp*U'           % pseudoinverse
Ap =

   -0.0503686   -0.0196560   -0.0073710    0.0294840    0.0479115
    0.8648649    0.4594595    0.2972973   -0.1891892   -0.4324324
```

Finally, we are prepared to solve the original system of equations. The least-squares solution is now simply $A^+\mathbf{y}$ (note that this is the matrix product A^+ times $\mathbf{y}$, not the sum $A+\mathbf{y}$).

```
>> Ap*y
ans =

   -1.5590
   46.3784
```

So, the correct linear equation is $y = -1.5590x + 46.3784$. Plot the original data and best-fitting line. The result is shown in Figure 5.2. □

5.5 Gram-Schmidt and the QR algorithm

5.5.1 The Gram-Schmidt process

Let $\mathbf{u}$ and $\mathbf{v}$ be two linearly independent vectors. Then the vector $\mathbf{u}-\text{proj}_{\mathbf{v}}(\mathbf{u})$ will be orthogonal to $\mathbf{v}$.

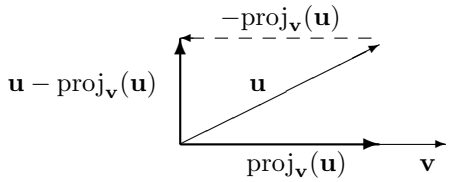


Figure 5.3: Orthogonal projection

Notice that the set $\{\mathbf{v}, \mathbf{u}-\text{proj}_{\mathbf{v}}(\mathbf{u})\}$ is now an orthogonal set which has the same span as the original set $\{\mathbf{v}, \mathbf{u}\}$. This use of orthogonal projections to make a linearly independent set into an orthogonal set is the basis of the famous *Gram-Schmidt process*.

Theorem 5.5.1. THE GRAM-SCHMIDT PROCESS
Let $\{\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_n\}$ be a linearly independent set. Then the following procedure will produce an orthogonal set $\{\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n\}$ with the same span.

$$\begin{aligned}
\mathbf{v}_1 &= \mathbf{u}_1 \\
\mathbf{v}_2 &= \mathbf{u}_2 - \text{proj}_{\mathbf{v}_1}(\mathbf{u}_2) \\
\mathbf{v}_3 &= \mathbf{u}_3 - \text{proj}_{\mathbf{v}_1}(\mathbf{u}_3) - \text{proj}_{\mathbf{v}_2}(\mathbf{u}_3) \\
&\vdots \\
\mathbf{v}_n &= \mathbf{u}_n - \text{proj}_{\mathbf{v}_1}(\mathbf{u}_n) - \text{proj}_{\mathbf{v}_2}(\mathbf{u}_n) - \cdots - \text{proj}_{\mathbf{v}_{n-1}}(\mathbf{u}_n)
\end{aligned}$$

To normalize, set

$$\mathbf{w}_i = \frac{\mathbf{v}_i}{\|\mathbf{v}_i\|}$$

Then the set $\{\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_n\}$ is an orthonormal set with the same span as $\{\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_n\}$ and $\{\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n\}$.

Example 5.5.2. Find an orthonormal set with the same span as

$$\{\langle 10, 9, -3, 0\rangle, \langle -7, 7, -3, 4\rangle, \langle 9, 1, -8, -1\rangle\}$$

Solution. Since we are going to make extensive use of vector projections, it would be a good idea to write a function that handles that part of the computation. This can be entered at the command line, or better yet, it can be saved in a function file `proj.m` and reused in future problems.

```
>> function vect = proj(u, v)
     vect = dot(u, v)/dot(v, v)*v;
   end
```

As defined, proj(u, v) now computes the projection of $\mathbf{u}$ onto $\mathbf{v}$.

Now, we will enter the original set of vectors as columns in a matrix U.

```
>> U = [10  9 -3  0; -7  7 -3  4;  9  1 -8 -1]'   % note transpose
U =

    10    -7     9
     9     7     1
    -3    -3    -8
     0     4    -1
```

Next, we go through the steps of the Gram-Schmidt process to create a matrix V whose columns are an orthogonal set with the same span as the original set.

```
>> V = zeros(4, 3);
>> V(:, 1) = U(:, 1);
>> V(:, 2) = U(:, 2) - proj(U(:, 2), V(:, 1));
>> V(:, 3) = U(:, 3) - proj(U(:, 3), V(:, 1)) - proj(U(:, 3), V(:, 2))
V =

    10.00000    -7.10526     0.37157
     9.00000     6.90526    -2.73222
    -3.00000    -2.96842    -6.95810
     0.00000     4.00000     0.21304
```

These vectors are orthogonal, but not yet unit vectors, so we normalize. The final output matrix W should have columns that are orthogonal unit vectors with the same span as the original set.

```
>> W = zeros(4, 3);
>> W(:, 1) = V(:, 1)/norm(V(:, 1));
>> W(:, 2) = V(:, 2)/norm(V(:, 2));
>> W(:, 3) = V(:, 3)/norm(V(:, 3))
W =

    0.72548   -0.64071    0.04962
    0.65293    0.62268   -0.36490
   -0.21764   -0.26768   -0.92929
    0.00000    0.36070    0.02845
```

The columns of W are the desired orthonormal set. □

We might want to verify that the process worked. As a spot check, we can look at the dot product of any two columns and we should get 0. Also, each column should have norm 1.

```
>> dot(W(:, 1), W(:, 3))
ans =    2.2204e-016

>> norm(W(:, 2))
ans =  1
```

The results are as expected, the usual minor round-off error in the dot product notwithstanding.

5.5.2 QR decomposition

We have already seen several important matrix factorizations. The Gram-Schmidt process is the key to another, one that turns out to provide a good means for finding eigenvalues numerically. This is known as the *QR decomposition*.

Theorem 5.5.3. *Let A be a nonsingular square matrix. Then there exists an orthogonal matrix Q and an upper triangular matrix R such that $A = QR$.*

Here's how to find Q and R.

1. Apply the Gram-Schmidt process to the columns of A. Use the resulting orthonormal vectors as columns of Q.
2. Let $R = \begin{bmatrix} \mathbf{q}_1 \cdot \mathbf{a}_1 & \mathbf{q}_1 \cdot \mathbf{a}_2 & \mathbf{q}_1 \cdot \mathbf{a}_3 & \cdots & \mathbf{q}_1 \cdot \mathbf{a}_n \\ 0 & \mathbf{q}_2 \cdot \mathbf{a}_2 & \mathbf{q}_2 \cdot \mathbf{a}_3 & \cdots & \mathbf{q}_2 \cdot \mathbf{a}_n \\ 0 & 0 & \mathbf{q}_3 \cdot \mathbf{a}_3 & \cdots & \mathbf{q}_3 \cdot \mathbf{a}_n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \mathbf{q}_n \cdot \mathbf{a}_n \end{bmatrix}$, where $\mathbf{q}_i$ is the ith column of Q and $\mathbf{a}_j$ is the jth column of A.

Example 5.5.4. Find the QR decomposition of the matrix $A = \begin{bmatrix} 5 & 7 & 0 \\ 10 & 8 & 0 \\ 5 & 6 & -5 \end{bmatrix}$.

Solution. First we apply the Gram-Schmidt process to A.

```
>> A = [5 7 0; 10 8 0; 5 6 -5]
A =

     5     7     0
    10     8     0
     5     6    -5

>> Q = zeros(3, 3);
>> Q(:, 1) = A(:, 1)/norm(A(:, 1));
>> Q(:, 2) = A(:, 2) - proj(A(:, 2), Q(:, 1));
>> Q(:, 2) = Q(:, 2)/norm(Q(:, 2));
>> Q(:, 3) = A(:, 3) - proj(A(:, 3), Q(:, 1)) - proj(A(:, 3), Q(:, 2));
>> Q(:, 3) = Q(:, 3)/norm(Q(:, 3))
Q =

    0.40825    0.72900    0.54944
    0.81650   -0.56077    0.13736
    0.40825    0.39254   -0.82416
```

Notice that we normalized each vector as we went through the process to find Q. Now, let's verify that Q is orthogonal. For an orthogonal matrix, $Q^{-1} = Q^T$, so a good way to check for orthogonality is to compute Q^TQ, which should be an identity matrix.

```
>> Q'*Q
ans =

    1.00000   -0.00000    0.00000
   -0.00000    1.00000   -0.00000
    0.00000   -0.00000    1.00000
```

This looks correct (some round-off error can be seen if we check more digits than displayed here). Now, we build R using the appropriate dot products of columns of Q and A.

```
>> R = zeros(3, 3);
>> R(1, 1) = dot(Q(:, 1), A(:, 1));
>> R(1, 2) = dot(Q(:, 1), A(:, 2));
>> R(1, 3) = dot(Q(:, 1), A(:, 3));
>> R(2, 2) = dot(Q(:, 2), A(:, 2));
>> R(2, 3) = dot(Q(:, 2), A(:, 3));
>> R(3, 3) = dot(Q(:, 3), A(:, 3))
R =

   12.24745   11.83920   -2.04124
    0.00000    2.97209   -1.96270
    0.00000    0.00000    4.12082
```

Of course, for a larger problem, we would use loops to compute the entries in R. Finally we check to see that $QR = A$.

```
>> Q*R
ans =

    5.00000    7.00000    0.00000
   10.00000    8.00000    0.00000
    5.00000    6.00000   -5.00000
```

It works as expected. □

5.5.3 The QR algorithm

The QR decomposition is the basis of a numerical method for finding eigenvalues.

Theorem 5.5.5. THE QR ALGORITHM

Let A be an $n \times n$ matrix with n real eigenvalues.

Set $A_1 = A$.

For each $k = 1, 2, 3, \ldots$ do the following:

(i) Find the QR decomposition of A_k, $A_k = Q_k R_k$.

(ii) Set $A_{k+1} = R_k Q_k$.

Repeat steps (i) and (ii).

As k increases, the matrices A_k approach an upper triangular form with the eigenvalues of A on the diagonal.

Example 5.5.6. Apply three iterations of the QR algorithm to the matrix $A = \begin{bmatrix} 5 & 7 & 0 \\ 10 & 8 & 0 \\ 5 & 6 & -5 \end{bmatrix}$.

Solution. We will use the built-in QR-decomposition function, [Q R] = qr(A).

```
>> A1 = A
A1 =

     5     7     0
    10     8     0
     5     6    -5

>> [Q1 R1] = qr(A1);
>> A2 = R1*Q1
A2 =

   13.8333   -1.4881   10.0378
   -1.6254   -2.4371   -2.0258
    1.6823   -1.6176   -3.3962

>> [Q2 R2] = qr(A2);
>> A3 = R2*Q2
A3 =

   15.159187    4.145301   -6.805968
   -0.013431   -4.054621    1.168669
    0.430485    1.750645   -3.104566

>> [Q3 R3] = qr(A3);
>> A4 = R3*Q3
A4 =

   14.959822    6.640881    5.216123
    0.065351   -4.860028   -0.375929
    0.064287   -0.846029   -2.099794
```

It turns out that the correct eigenvalues of A are 15, -5, and -2. These values are already evident on the diagonal after only three iterations. □

It is a simple matter to codify the algorithm into a loop, which allows easily running a large number of iterations. This is left as an exercise for the reader (see Exercise 11).

Chapter 5 Exercises

1. Let $A = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 1 & -1 \\ 0 & 2 & 4 \end{bmatrix}$, $B = \begin{bmatrix} 2 & -2 & 1 \\ 1 & -1 & 1 \\ -3 & 2 & -2 \end{bmatrix}$, and $C = \begin{bmatrix} 1 & -1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 2 \end{bmatrix}$.

   For each matrix, do the following:

   (a) Find the eigenvalues and eigenvectors by hand. First give a parametric description for the set of eigenvectors for each eigenvalue, then choose representative eigenvectors with integer (or Gaussian/complex integer) components for each eigenvalue.

   (b) Use Octave to find the eigenvalues and eigenvectors. Compare the Octave solution to your by hand solution.

   (c) How many linearly independent eigenvectors does each matrix have?

2. Suppose a hypothetical state is divided into four regions, A, B, C, and D. Each year, a certain number of people will move from one region to another, changing the population distribution. The initial populations are given below:

Region	Population
A	719
B	910
C	772
D	807

   The following table records how the population moved in one year.

		To A	B	C	D
From	A	624	79	2	14
	B	79	670	70	91
	C	52	6	623	91
	D	77	20	58	652

   For example, we see that A began with $624 + 79 + 2 + 14 = 719$ residents. Of these, 624 stayed in A, 79 moved to B, 2 moved to C, and 14 moved to D. From this empirical data, we can give approximate probabilities for moving from A. Of the 719 residents, 624 stayed in A, so the probability of "moving" from A to A is $624/719 = 0.8678720$. The probability of moving from A to B is $79/719 = 0.1098748$, and so on.

   (a) Find the transition matrix T for this Markov chain. This is done by converting each entry in the table above to a probability, then transposing.

   (b) Express the initial population distribution as a probability vector $\mathbf{x}$. Remember, the components must add to 1.

   (c) Find the population distribution (expressed as percentages) in 5 years and in 10 years.

   (d) Compute the eigenvalues and eigenvectors for T and use the eigenvector for $\lambda = 1$ to construct an equilibrium vector $\mathbf{q}$ for this Markov chain. This represents a population distribution for which there is no further change from year to year. Verify that the distribution is in equilibrium by computing several future states, such as $T^{25}\mathbf{q}$ and $T^{50}\mathbf{q}$. Is there any change in the distribution?

3. Refer to the random walk Markov chain from Example 5.2.1. Set up the transition matrices for the following modified scenarios and find an equilibrium vector for each case.

   (a) At intersections 2, 3, or 4, move to the left or right with equal probability. At intersections 1 or 5, move back to where you came from.

   (b) At intersections 2, 3, or 4, move to the left with probability 0.4 or to the right with probability 0.6. At intersections 1 or 5, either stop or move back to where you came from with equal probability.

4. Which of the matrices in Exercise 1 can be diagonalized? For each matrix, give a diagonalization if possible, or explain why no diagonalization is possible.

5. Diagonalize the matrix $A = \begin{bmatrix} 1 & 4 \\ 1 & -2 \end{bmatrix}$ as $A = S\Lambda S^{-1}$ and use this to calculate A^{50}. Show all the steps needed to find the eigenvalues, eigenvectors, etc.

6. Orthogonally diagonalize each symmetric matrix. Verify that the matrix equals $Q\Lambda Q^T$ and show that Q is orthogonal by verifying that $QQ^T = Q^TQ = I$.

$$A = \begin{bmatrix} 1 & -2 \\ -2 & 0 \end{bmatrix} \text{ and } B = \begin{bmatrix} 1 & 1 & 3 \\ 1 & 1 & 3 \\ 3 & 3 & 9 \end{bmatrix}$$

   Solve by hand first, then check your work with Octave. (Note that B requires the Gram-Schmidt process.)

7. Find the SVD of the matrix $\begin{bmatrix} 2 & 3 \\ 0 & 2 \end{bmatrix}$ without using the svd command. Show all the steps needed to find the eigenvalues, eigenvectors, etc. Verify that $A = U\Sigma V^T$.

8. Use the pseudoinverse to find the least-squares line $y = ax + b$ through the given set of points.

$$\{(-1, 5), (1, 4), (2, 2.5), (3, 0)\}$$

   You may use the svd command, but show all the rest of the details, including construction of the pseudoinverse. Include a plot of the data values and the least-squares line.

9. Write an Octave script that takes a matrix U with linearly independent columns and outputs a matrix V with orthonormal columns. The core loop could look like this (or use your own formulation):

```
V(:, 1) = U(:, 1)/norm(U(:, 1));
for i = 2:n
  V(:, i) = U(:, i);
  for j = 1:i-1
    V(:, i) = V(:, i) - proj(U(:, i), V(:, j));
  end
  V(:, i) = V(:, i)/norm(V(:, i));
end
```

   You will need to determine m and n and from the dimensions of U and the function proj(u,v) must be defined. Test your code on the vectors from Example 5.5.2.

10. Use your code from Exercise 9 as the starting point of a user-defined function, stored in a function file, that computes the QR decomposition of an $n \times n$ matrix A. Test your function on a randomly generated 4×4 matrix, A = rand(4, 4). Check Q for orthogonality by computing $Q^T Q$, which should be an identity matrix, and verify that $A = QR$.

11. Using Octave's built-in [Q R] = qr(A) function for the QR decomposition, write a script to approximate the eigenvalues of the matrix

$$A = \begin{bmatrix} 1 & -1 & 2 \\ -1 & 1 & -2 \\ 2 & -2 & 0 \end{bmatrix}$$

Run your loop through ten iterations. The actual eigenvalues are integers. Were you able to determine the correct values from the QR algorithm?

Chapter 6

Multivariable calculus and differential equations

6.1 Space curves

Plotting a curve in 3-dimensions is similar to the 2-dimensional plotting explained in Section 1.4. To plot space curves, we use the command plot3(x, y, z), where x, y, and z correspond to the parametric equations for the function. For example, let's plot a simple helix, with vector equation $\mathbf{r}(t) = \sin(t)\mathbf{i} + \cos(t)\mathbf{j} + t\mathbf{k}$. First we generate a row vector for the parameter t, then we calculate the range for x, y, and z.

```
>> t = linspace(0, 2*pi, 30);
>> x = sin(t);
>> y = cos(t);
>> z = t;
>> plot3(x, y, z)
```

The graph is shown in Figure 6.1.

Now consider a more complicated curve, like

$$x = (5 + \sin 25t)\cos t,\ y = (5 + \sin 25t)\sin t,\ z = \cos 25t$$

This is a "toroidal spiral." We will need to use a much finer increment for t to get a smooth picture.

```
>> t = linspace(0, 2*pi, 500);
>> x = (5 + sin(25*t)).*cos(t);     % elementwise product
>> y = (5 + sin(25*t)).*sin(t);     % elementwise product
>> z = cos(25*t);
>> plot3(x, y, z)
```

These types of graphs are not easy to draw without a computer! See Figure 6.2. Note the elementwise operations, which we utilize throughout this chapter (see Section 1.4.1).

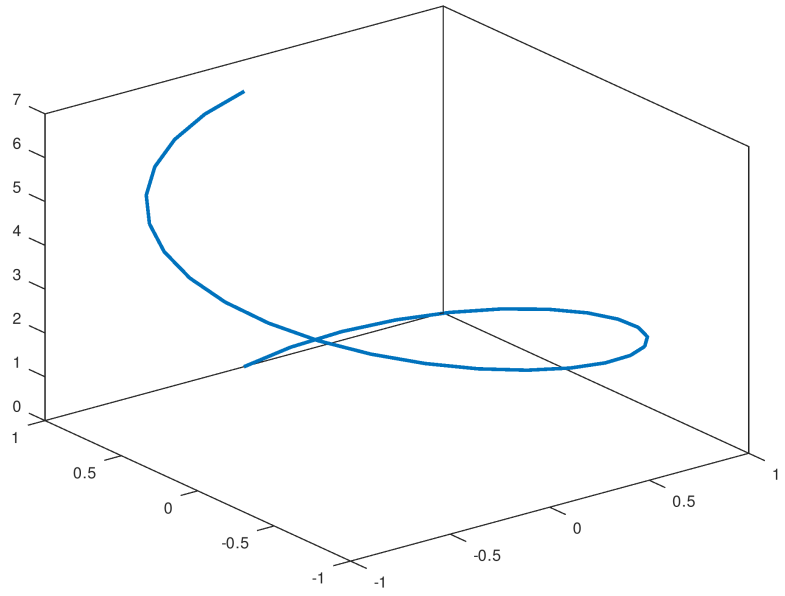


Figure 6.1: Helix

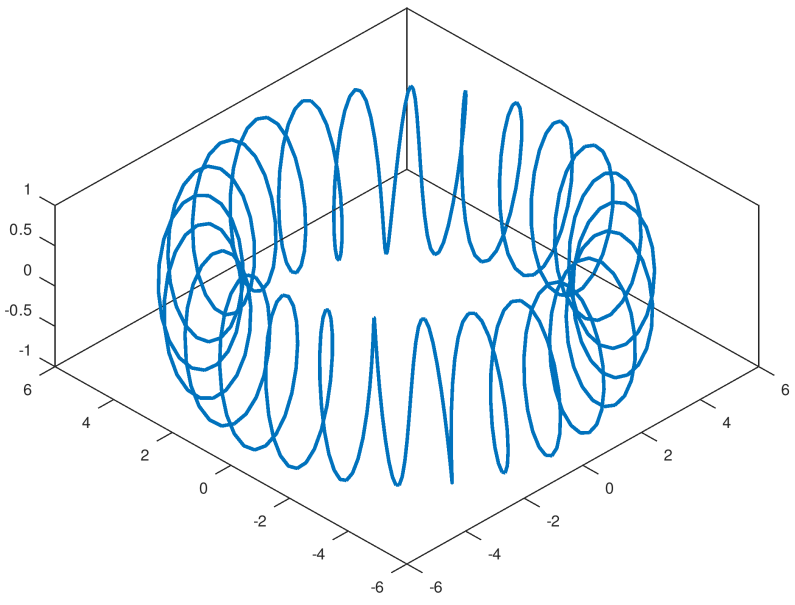


Figure 6.2: Toroidal spiral

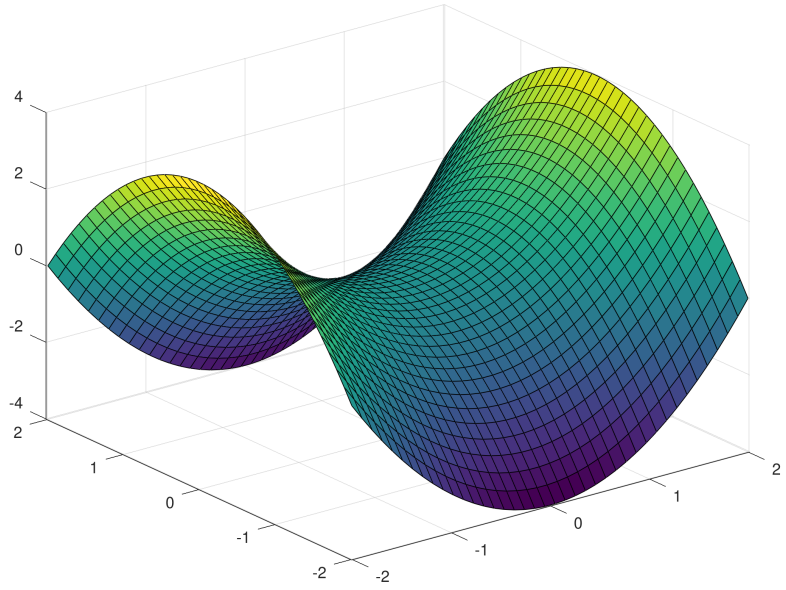


Figure 6.3: Saddle surface

6.2 Surfaces

How about plotting surfaces rather than curves? In this case, we use a two-dimensional "mesh" of input values and calculate the range using a function of two variables. For example, let's graph the familiar saddle surface shown in Figure 6.3, defined by $f(x, y) = x^2 - y^2$.

First we define the domain over which the function will be plotted.

```
>> x = linspace(-2, 2, 40);
>> y = linspace(-2, 2, 40);
```

Then, we use the meshgrid command to create a mesh of all possible combinations of x and y in the domain. We will adopt the useful convention of distinguishing the meshgrid variables from their linear counterparts by naming them with corresponding capital letters, X and Y.

```
>> [X Y] = meshgrid(x, y);
```

Now calculate the range using these meshgrid variables.

```
>> Z = X.^2 - Y.^2;
```

Finally, we can plot the surface with the surf command.

```
>> surf(X, Y, Z)
```

The default surface graph is color coded by elevation; type help colormap for a list of alternative color schemes. The graph can be rotated in space by clicking the rotate icon on the graph window

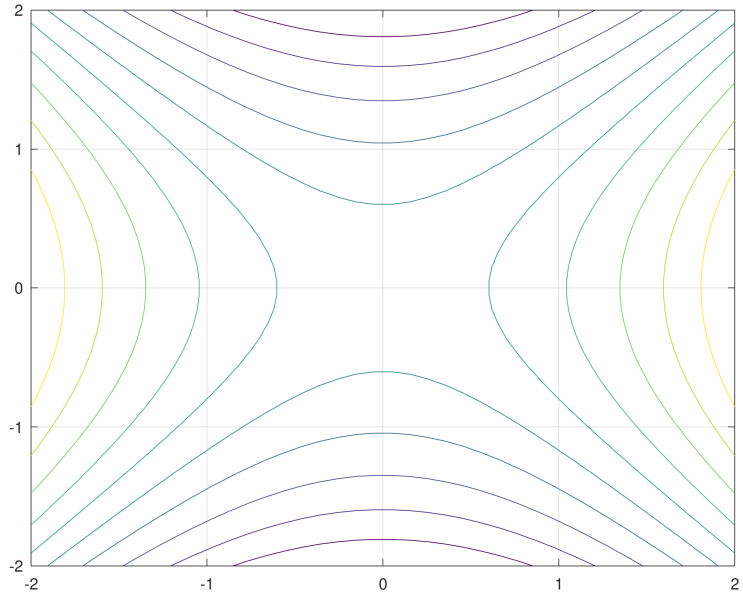


Figure 6.4: Contour plot of saddle surface

toolbar. To see a graph of the mesh without the surface filled in, use mesh(X, Y, Z). Also try contour(X, Y, Z) to see a contour plot of the surface, as shown in Figure 6.4.

Example 6.2.1. Graph the surface $f(x, y) = (1 + xy)(x + y)$.

Solution. We begin using the same basic procedure outlined above, this time choosing to plot the function over $[-5, 5] \times [-5, 5]$.

```
>> % define the domain
>> x = linspace(-5, 5, 30);
>> y = linspace(-5, 5, 30);
>> [X Y] = meshgrid(x, y);

>> % calculate the range
>> Z = (1 + X.*Y).*(X + Y);

>> % plot the surface
>> surf(X, Y, Z)
```

The graph as shown in Figure 6.5 appears unremarkable. However, an analysis of the partial derivatives shows the existence of two critical points, at $(1, -1)$ and $(-1, 1)$, each of which corresponds to a saddle point (the reader should verify this). To see these features clearly, we need to manually adjust the viewpoint.

```
>> % manually set new axis limits
>> axis([-5 5 -5 5 -10 10])
```

The two saddle points are now apparent (Figure 6.6). Notice that the axis command takes a 6-element vector as its argument, of the form [Xmin Xmax Ymin Ymax Zmin Zmax]. □

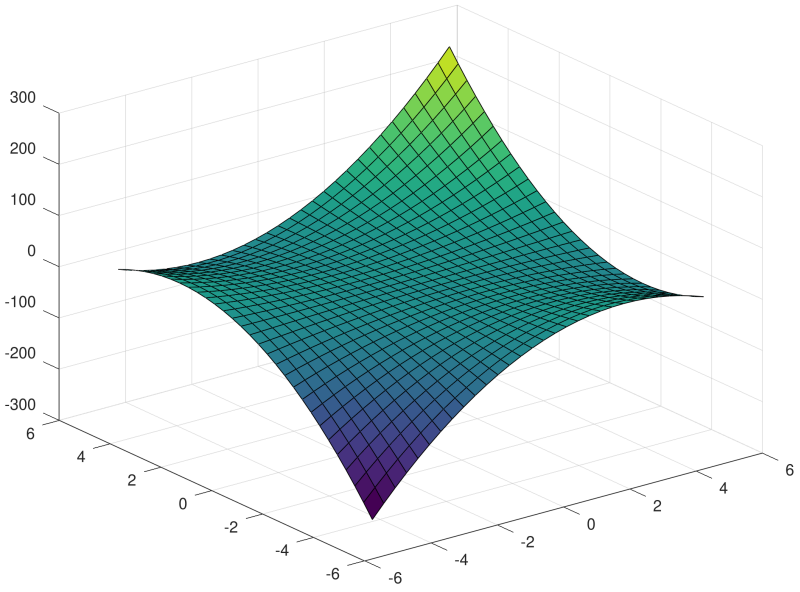


Figure 6.5: Graph of $f(x, y) = (1 + xy)(x + y)$

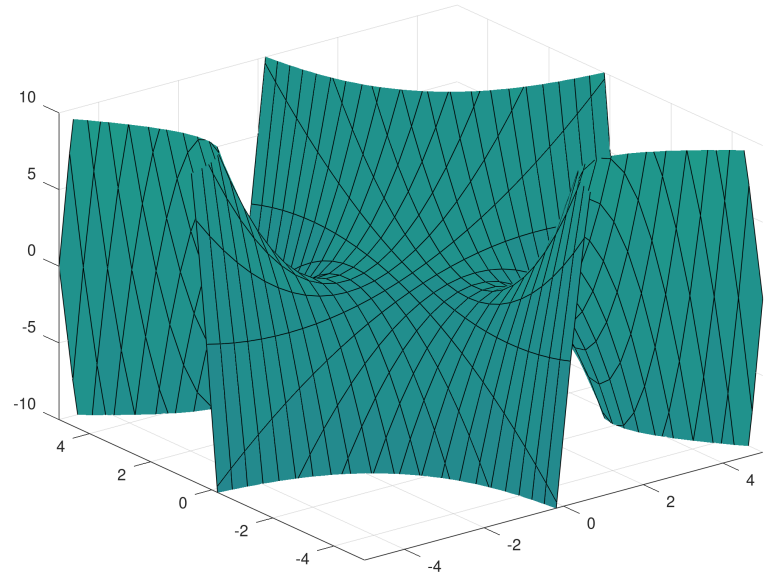


Figure 6.6: Revised view showing two saddle points

6.2.1 Change of variables

To plot a surface defined over a restricted, nonrectangular domain it may be necessary to make a change of variables. Typical examples include surfaces defined in terms of polar/cylindrical or spherical coordinates, but more general changes of variable are also possible.

Example 6.2.2. Graph the surface $f(x, y) = \sqrt{9 - x^2 - y^2}$.

Solution. The function corresponds to the upper half of a radius-3 sphere. If we naively attempt to plot the function over $[-3, 3] \times [-3, 3]$, we will run into trouble:

```
>> x = linspace(-3, 3, 30);
>> y = x;
>> [X Y] = meshgrid(x, y);
>> Z = sqrt(9 - X.^2 - Y.^2);
>> surf(X, Y, Z)
error: mesh: X, Y, Z arguments must be real
error: called from
    surface>__surface__ at line 123 column 9
    surface at line 63 column 19
    surf at line 72 column 10
```

Of course the problem is that over parts of the rectangular region of interest, the function is undefined (or more precisely, the function values are imaginary). A quick-and-dirty workaround is to simply plot the real part of Z. This gives a satisfactory result in this case, but in general, this method is less than ideal.

```
>> surf(X, Y, real(Z))
```

The graph restricted to the real component of the function is shown in Figure 6.7.

A better approach would be to graph the function using polar/cylindrical coordinates. To do so, we create an $r\theta$-meshgrid, with $0 \le r \le 3$ and $0 \le \theta \le 2\pi$.

```
>> r = linspace(0, 3, 25);
>> theta = linspace(0, 2*pi, 25);
>> [R T] = meshgrid(r, theta);
```

In terms of cylindrical coordinates, $z = \sqrt{9 - r^2}$.

```
>> Z = sqrt(9 - R.^2);
```

Now, calculate x and y using the standard polar to rectangular identities, $x = r\cos(\theta)$, $y = r\sin(\theta)$. Be sure to use the meshgrid variables.

```
>> X = R.*cos(T);
>> Y = R.*sin(T);
```

Finally, graph the surface.

```
>> surf(X, Y, Z)
```

The improved graph of the hemisphere is shown in Figure 6.8. □

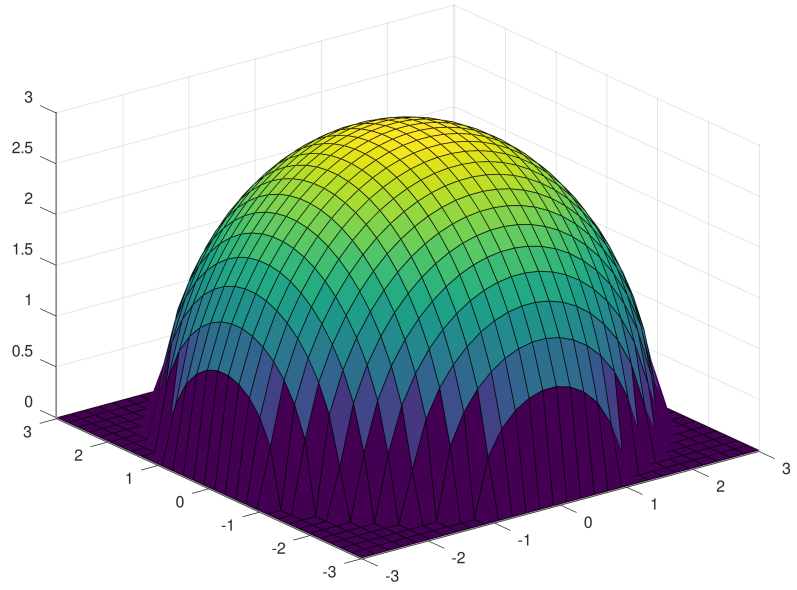


Figure 6.7: Graph showing real part of a radius-3 hemisphere

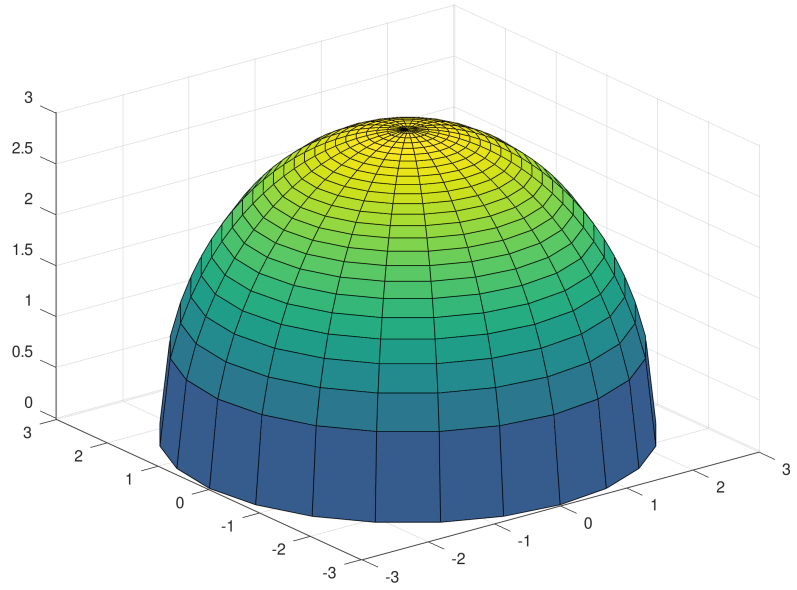


Figure 6.8: Radius-3 hemisphere graphed in polar form

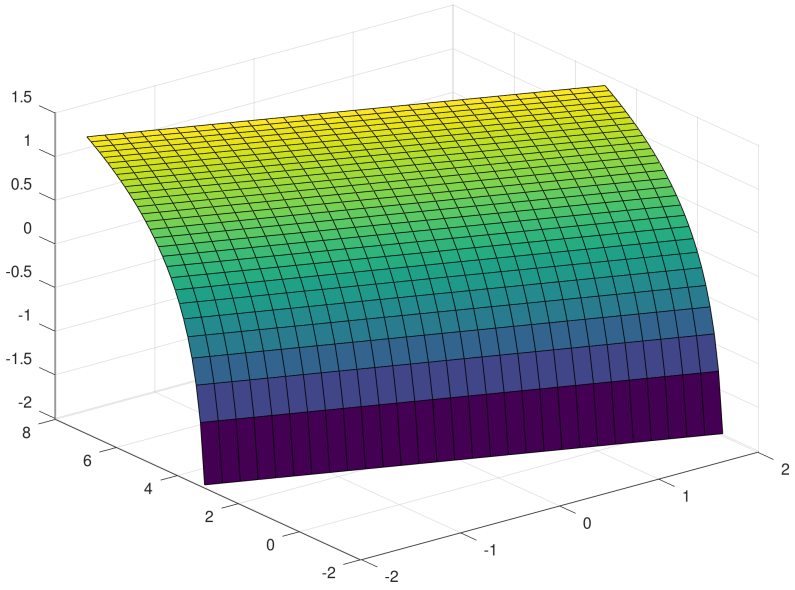


Figure 6.9: Graph of $f(x, y) = \ln(x + y - 1)$ using a change of variables

Example 6.2.3. Graph the function $f(x) = \ln(x + y - 1)$.

Solution. This function is defined only on the half-plane $x+y > 1$. To graph such a function in Octave we must make a suitable change of variables. In this case, the domain of the function suggests the substitution $u = x+y$. We first create a ux-meshgrid, where $u > 1$. Then, using this change of variables, $z = \ln(u - 1)$ and $y = u - x$.

```
>> % define the ux-mesh
>> u = linspace(1, 5, 30);
>> x = linspace(-2, 2, 30);
>> [U X] = meshgrid(u, x);

>> % calculate y and z using the change of variables substitution
>> Z = log(U - 1);
>> Y = U - X;

>> % plot the surface
>> surf(X, Y, Z)
```

The resulting graph is shown in Figure 6.9. □

Example 6.2.4. The function

$$\rho = 1 + \frac{1}{4}\sin(5\phi)\cos(6\theta), 0 \leq \theta \leq 2\pi, 0 \leq \phi \leq \pi$$

in spherical coordinates is known as a bumpy sphere. Graph this function.

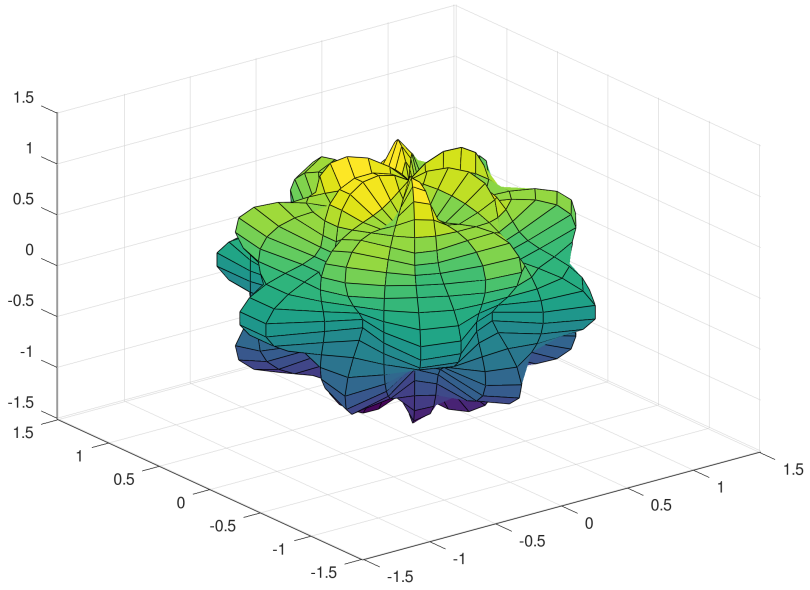


Figure 6.10: Bumpy sphere

Solution. We use a $\theta\phi$-meshgrid to calculate ρ. Then we can calculate x, y, and z using the standard spherical to rectangular coordinate identities.

```
>> % define phi (P) and theta (T)
>> theta = linspace(0, 2*pi, 30);
>> phi = linspace(0, pi, 30);
>> [T P] = meshgrid(theta, phi);

>> % calculate rho (R)
>> R = 1 + 1/4*sin(5*P).*cos(6*T);

>> % use spherical identities for X, Y, Z
>> X = R.*sin(P).*cos(T);
>> Y = R.*sin(P).*sin(T);
>> Z = R.*cos(P);

>> % plot the surface
>> surf(X, Y, Z)
```

The graph is shown in Figure 6.10. □

6.2.2 Parametric surfaces

Some surfaces are most easily described parametrically. We can graph such a surface in Octave by generating meshgrid arrays for the parameters, then calculating x, y, and z.

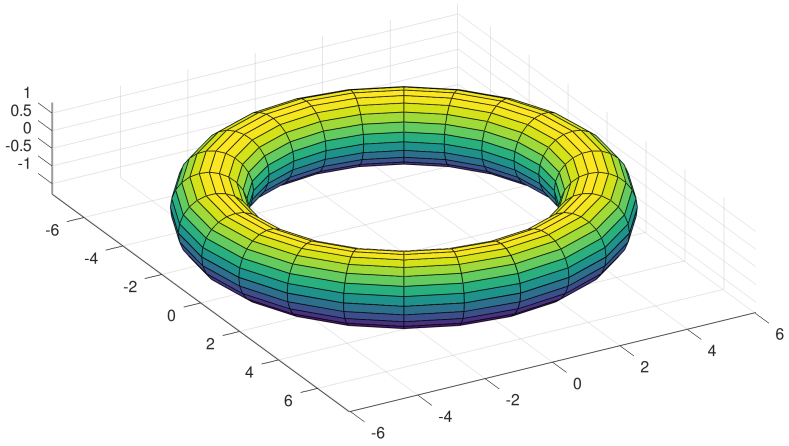


Figure 6.11: Torus

Example 6.2.5. The curve in Figure 6.2 lies on the surface of a torus, defined parametrically as

$$\begin{aligned} x &= (5+\cos(u))\cos(v) \\ y &= (5+\cos(u))\sin(v) \\ z &= \sin(u), \end{aligned}$$

where $u, v \in [0, 2\pi]$. Graph this surface.

Solution. We begin by defining the parameters.

```
>> u = linspace(0, 2*pi, 25);
>> v = u;
>> [U V] = meshgrid(u, v);
```

Calculate x, y, and z:

```
>> X = (5 + cos(U)).*cos(V);
>> Y = (5 + cos(U)).*sin(V);
>> Z = sin(U);
```

Now, plot the surface.

```
>> surf(X, Y, Z)
>> axis('equal')
```

The result is shown in Figure 6.11. Note the use of axis('equal') to force an equal aspect ratio. □

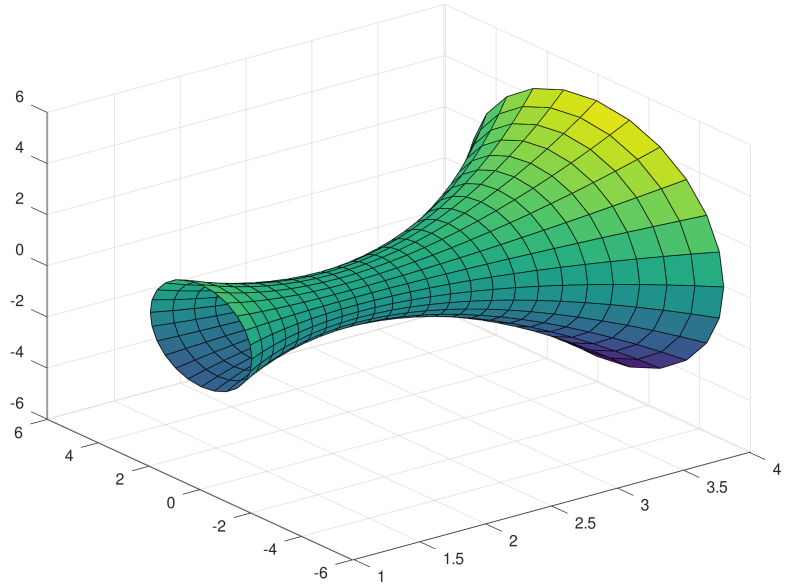


Figure 6.12: Solid of revolution

Solids of revolution can also be graphed as parametrically defined surfaces. For example, parametric equations for the surface formed by rotating the graph of $y = f(x)$ about the x-axis are

$$x = x \tag{6.1}$$
$$y = f(x)\cos(t) \tag{6.2}$$
$$z = f(x)\sin(t) \tag{6.3}$$

where $0 \leq t \leq 2\pi$ and $a \leq x \leq b$.

Equations 6.1–6.3 can be modified as needed to produce rotations around the other axes.

Example 6.2.6. Graph the solid obtained by rotating $f(x) = x^2 - 4x + 5$, for $1 \leq x \leq 4$, about the x-axis.

Solution. These commands will graph the surface.

```
>> x = linspace(1, 4, 25);      % define the domain
>> f = @(x) x.^2 - 4*x + 5;     % define the function
>> t = linspace(0, 2*pi, 25);   % define the parameter
>> [X T] = meshgrid(x, t);      % xt-mesh
>> Y = f(X).*cos(T);            % calculate Y
>> Z = f(X).*sin(T);            % calculate Z
>> surf(X, Y, Z)                % graph surface
```

The result is in Figure 6.12. □

6.3 Multiple integrals

We showed methods for evaluating single integrals numerically in Chapter 3. We now consider multiple integrals. The commands dblquad and triplequad can be used to evaluate double and triple integrals over a rectangle or rectangular box.

For example, let's evaluate:

$$\int_{-1}^{3}\int_{0}^{2}(x^2y+2y)\,dx\,dy$$

```
>> % double integral using dblquad
>> function z = f(x, y)
     z = x.^2.*y + 2*y;
   end
>> dblquad('f', 0, 2, -1, 3)
ans =  26.667
```

Evaluating over a nonrectangular domain is a trickier problem. Let's give it a try.

Example 6.3.1. Evaluate

$$\iint_R (x^2y+y^2x)\,dA$$

over the region R bounded by the graphs of $y=x^2$ and $y=\sqrt{x}$.

Solution. An analysis of the region of integration (Figure 6.13) shows that we need to evaluate the following iterated integral:

$$\int_{0}^{1}\int_{x^2}^{\sqrt{x}}(x^2y+y^2x)\,dy\,dx$$

We need to evaluate over only part of the rectangle $[0,1]\times[0,1]$. One approach is to define the integrand to be 0 for values outside of the region of integration. We do this using *logical functions*. Logical functions simply test whether a statement is true and return a value of 1 if true or 0 if false. For example $2+3<4$ returns 0, since the inequality is false. We can also use Boolean operators, like and (&) and or (|). Our region demands that we meet two conditions, $y>x^2$ and $y<\sqrt{x}$, so we use these conditions to define the function. By multiplying the integrand by the correct logical operator, it is set to 0 outside the region of interest.

```
>> % double integral over a nonrectangular domain
>> function z = f(x, y)
     z = (x.^2.*y + y.^2.*x) .* ((y > x.^2) & (y < sqrt(x)));
   end
>> dblquad('f', 0, 1, 0, 1)
ans =  0.10701
```

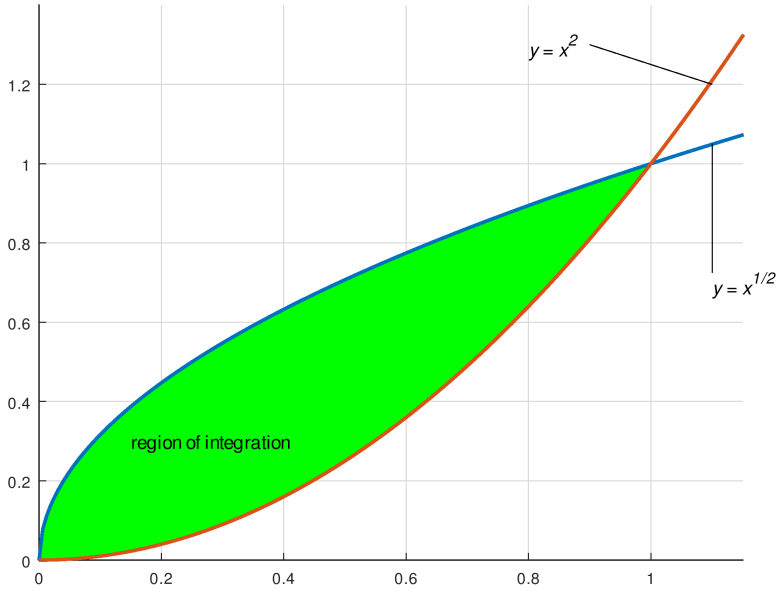


Figure 6.13: Region of integration for Example 6.3.1

Thus $\int_0^1 \int_{x^2}^{\sqrt{x}} (x^2 y + y^2 x)\, dx\, dy \approx 0.10701$. This is reasonably close to the exact value of $3/28$, but not in perfect agreement. The problem is that we have defined f as a discontinuous function (see Figure 6.14), but the quadrature algorithm works best on a smooth integrand. □

Besides its relative simplicity, the approach in Example 6.3.1 is nice for two further reasons: It illustrates the formal definition of a double Riemann integral over a nonrectangular domain (see [7, §5.2]), and it also allows us to plot the surface over the region of interest (Figure 6.14).

```
>> x = linspace(0, 1, 30);
>> y = x;
>> [X Y] = meshgrid(x, y);
>> Z = f(X, Y);
>> surf(X, Y, Z)
```

If we are unsatisfied with the numerical accuracy of this method for the double integral, another approach is to use a change of variables to transform to a rectangular region of integration as follows:

$$y = y_1 + u(y_2 - y_1) \tag{6.4}$$
$$dy = (y_2 - y_1)\, du \tag{6.5}$$

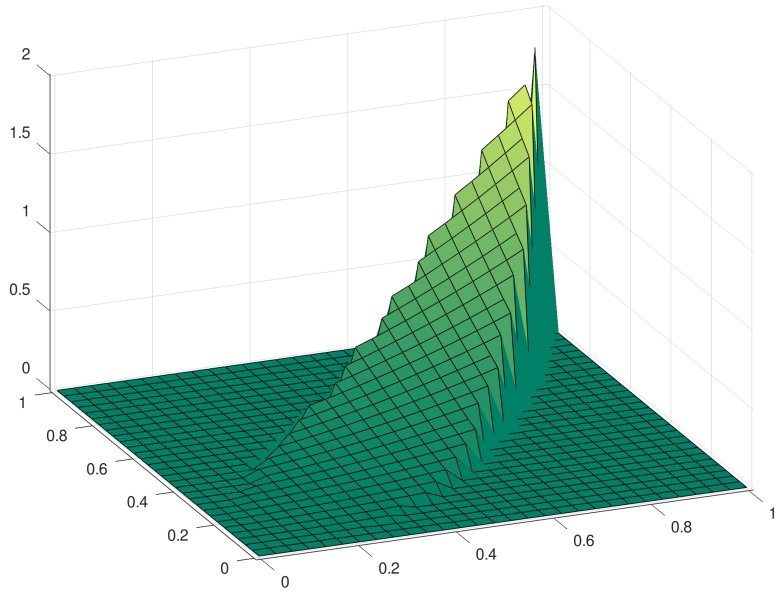


Figure 6.14: Solid volume for Example 6.3.1

Then as y ranges from y_1 to y_2, u ranges from 0 to 1 and the integrand becomes

$$\int_{x_1}^{x_2}\int_{y_1}^{y_2} f(x,y)\,dy\,dx = \int_{x_1}^{x_2}\int_{0}^{1} f(x, y_1 + u(y_2 - y_1))(y_2 - y_1)\,du\,dx \tag{6.6}$$

This is a bit cumbersome to enter in Octave. We'll use a series of *anonymous functions* (see page 39) to define the integrand, then try dblquad again.

Example 6.3.2. Use the change of variable formulas in Equations 6.4–6.6 to evaluate

$$\int_0^1 \int_{x^2}^{\sqrt{x}} (x^2 y + y^2 x)\,dy\,dx$$

Solution.

```
>> f1 = @(x, y) x.^2.*y + y.^2.*x;
>> y1 = @(x) x.^2;
>> y2 = @(x) sqrt(x);
>> f2 = @(x, u) f1(x, y1(x) + u.*(y2(x) - y1(x))) .* (y2(x) - y1(x));
>> format long                % display additional decimal places
>> dblquad(f2, 0, 1, 0, 1) % no quotes around anonymous function name
ans =  0.107142857143983
>> 3/28                       % compare result to known exact answer
ans =  0.107142857142857
```

This approach gives a more satisfactory result. □

If one wishes to evaluate many integrals of this form, writing a *function file* to automate the above steps would be a good idea.

Example 6.3.3. Write an Octave function file that computes

$$\iint_R f(x,y)\,dA$$

over the region R bounded by the graphs of $y = y_1(x)$, $y = y_2(x), x = a,$ and $x = b$, using the change of variables in Equations 6.4–6.6.

Solution. A function file is similar to a script; it is a plain text .m-file containing a series of Octave commands. To be recognized as a function file, the first line of code (excluding comments and white space) must be function. With the file placed in the load path, it can then be run from the command line like any other Octave function. The function name should match the file name. A well-written function file will include details like help text and provisions for error checking. Refer to [3]. We will give a minimal example that accomplishes our change of variables procedure. Use the text editor to enter the following code:

Octave Script 6.1: Double integral function file

```
% function file 'dblint.m'
% evaluates dblquad(f, x1, x2, y1, y2)
%   where f is an anonymous function of x and y
%   y1 and y2 are anonymous functions of x
%   x1 and x2 are real numbers

function val = dblint(f, x1, x2, y1, y2)
   f2 = @(x, u) f(x, y1(x) + u.*(y2(x) - y1(x))) .* (y2(x) - y1(x));
   val = dblquad(f2, x1, x2, 0, 1);
end
```

Note that the comment lines at the top of our function file will be displayed if we type help dblint. Thus we should strive to put a good description of the syntax in those lines. Now, to use this function, saved in our working directory as `dblint.m`, we need to define the integrand and the functions representing the limits of integration on y. Then we pass these to our function dblint. Let's try it on the integral from Example 6.3.1.

```
>> f = @(x, y) x.^2.*y + y.^2.*x;
>> y1 = @(x) x.^2;
>> y2 = @(x) sqrt(x);
>> dblint(f, 0, 1, y1, y2)
ans =  0.10714
```

It works! □

6.3.1 Double Riemann sums

We have seen a little about how Octave's built-in multiple integration functions work. Now suppose that instead we want to write our own algorithms. It is straightforward to write an

Octave script that will estimate a double integral over a rectangle using a double Riemann sum, taking sample points to be, say, the upper right hand corners of the subrectangles in the partition.

For our example, we'll use the function $f(x, y) = x + 2y^2$, defined on $R = [0, 2] \times [0, 4]$, using $m = n = 1000$. Use the text editor to enter the following code. Save the file with the name `dbl_Rsum.m` in your current working directory.

Octave Script 6.2: Nested loop double integral

```
% file 'dbl_Rsum.m'
% approximates a double integral using upper right hand corners of
%   subrectangles as sample points
%   --nested loop version

% define region of integration
a = 0;
b = 2;
c = 0;
d = 4;

% define function
function z = f(x, y)
  z = x + 2*y.^2;
end

% define partition
m = 1000
n = 1000

% calculate dA and initialize Riemann sum total
dx = (b - a)/m
dy = (d - c)/n
dA = dx*dy;
rsum = 0;

% compute double Riemann sum
for i = 1:m
  for j = 1:n
    rsum = rsum + dA *f(a + dx*i, c + dy*j);
  end
end

% display result
rsum
```

Now, run the script:

```
>> dbl_Rsum
m =  1000
n =  1000
dx =  0.0020000
dy =  0.0040000
rsum =  93.469
```

The estimate is reasonably close to the correct value of 93.333. However, the script is very slow, due to the inefficiency of running the calculation via nested loops. Notice that the program needs to compute one million function values in this example!

The routine can be sped up dramatically by using vectorized code, which takes advantage of Octave's fast algorithms for executing matrix and vector calculations. The new strategy is to generate a meshgrid array of the sample points, which is then used to define an $m \times n$ matrix containing the function values at the sample points. Finally, the Riemann sum is simply dA times the sum of all entries in the matrix. The vectorized script is shown below.

Octave Script 6.3: Vectorized double integral

```
% file 'dbl_Rsum_v2.m'
% approximates a double integral using upper right hand corners of
%   subrectangles as sample points
%   --vectorized version

% define region of integration
a = 0;
b = 2;
c = 0;
d = 4;

% define function
function z = f(x, y)
  z = x + 2*y.^2;
end

% define partition
m = 1000
n = 1000

% calculate dA
dx = (b - a)/m
dy = (d - c)/n
dA = dx*dy;

% calculate x and y values in partition
x = [a + dx : dx : b];
y = [c + dy : dy : d];

% create matrix of function values
[X Y] = meshgrid(x, y);
A = f(X, Y);

% calculate Reimann sum
rsum = dA*sum(sum(A))
```

This version gives the same result and executes significantly faster (try it!). But it is still not particularly accurate, considering the rather large values for m and n. The problem is that taking the upper right hand corners as sample points generally does not give the best estimate. The code can easily be improved by taking sample points at the midpoints of each rectangle. This minor adjustment is left as an exercise for the reader (see Exercise 10).

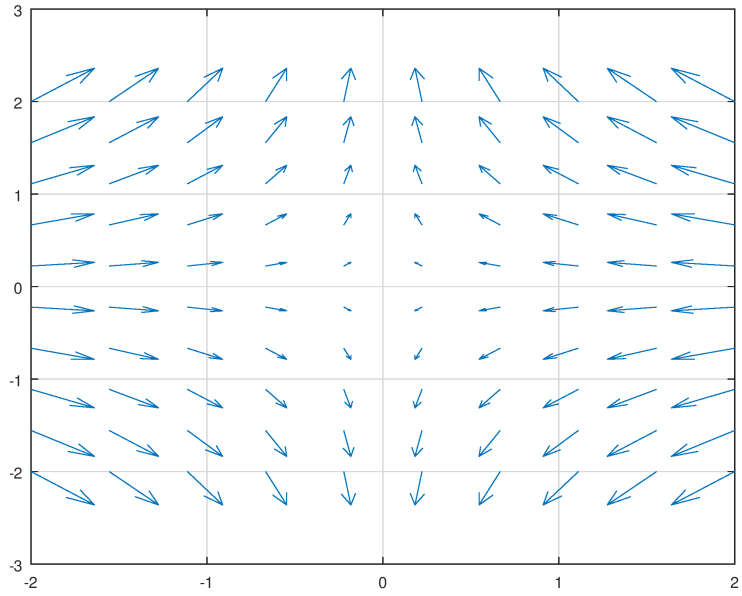


Figure 6.15: Vector field plot

6.4 Vector fields

A *vector field* assigns vectors to points in space. Vector fields are used to describe things like wind speed, fluid flow, electric charge, or gravitational force. A vector field is conveniently visualized by drawing a directed line segment for a series of representative points in the space. As any archer knows, a collection of arrows is called a *quiver*. Thus the command for plotting a vector field is quiver. The simplest form of the command is quiver(X, Y, U, V), where X and Y are meshgrid variables over which the field is plotted, and U and V are the x- and y-component functions, respectively.

Example 6.4.1. Graph the vector field $\mathbf{F}(x, y) = \langle -x, y \rangle$.

Solution.

```
>> x = linspace(-2, 2, 10);
>> y = x;
>> [X Y] = meshgrid(x, y);
>> quiver(X, Y, -X, Y);
>> grid on
```

See Figure 6.15. Some experimentation may be needed to determine the correct grid spacing. Too many points will result in an array of vectors too dense to interpret. □

We can also plot vector fields in three dimensions with quiver3 or add a vector field plot to the contour graph of a surface. We will illustrate these ideas with two more examples.

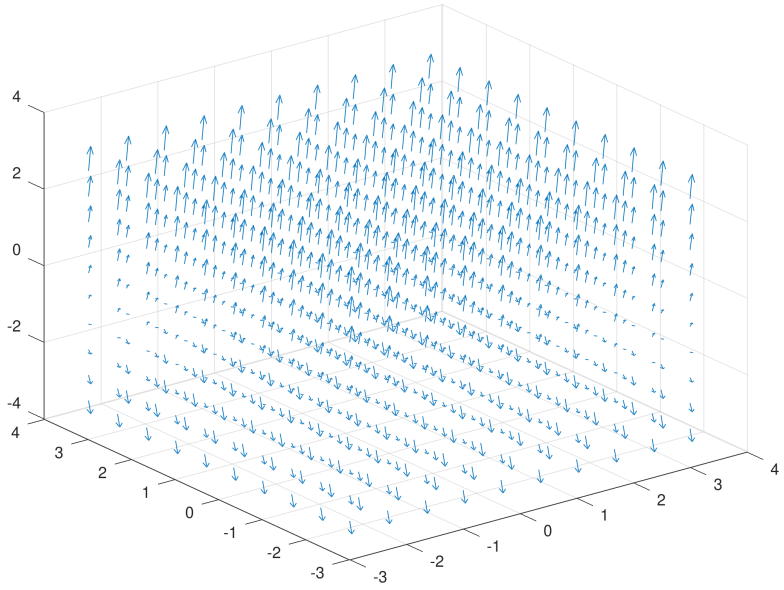


Figure 6.16: Three dimensional vector field

Example 6.4.2. Plot the vector field $\mathbf{F}(x, y, z) = \langle 1, 1, z \rangle$.

Solution.

```
>> x = linspace(-3, 3, 10);
>> y = x;
>> z = x;
>> [X Y Z] = meshgrid(x, y, z);
>> quiver3(X, Y, Z, ones(size(X)), ones(size(Y)), Z)
```

Note the use of the ones command to produce the constant terms. The result is in Figure 6.16. □

Example 6.4.3. Graph a contour plot of the Octave function "peaks" and its gradient field.

Solution. The command peaks plots an example graph of a surface with a number of maximums and minimums. Type help peaks for details, or just peaks to see the graph (Figure 6.17). It will be instructive to see its contours plotted together with its gradient field. We can use the built-in gradient function for this.

```
>> [X Y Z] = peaks;
>> [DX DY] = gradient(Z);
>> contour(X, Y, Z)
>> hold on
>> quiver(X, Y, DX, DY)
>> axis([-2 2 -2 2])
>> hold off
```

See the results in Figure 6.18. □

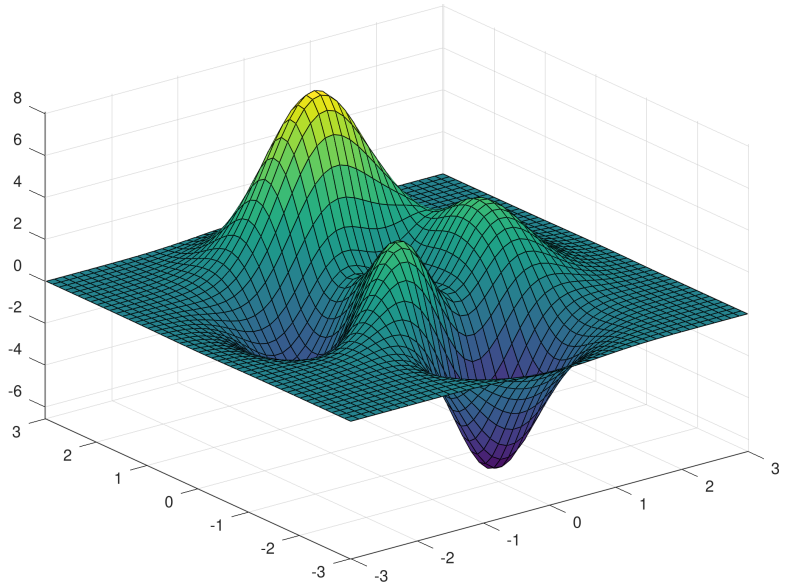


Figure 6.17: The "peaks" surface

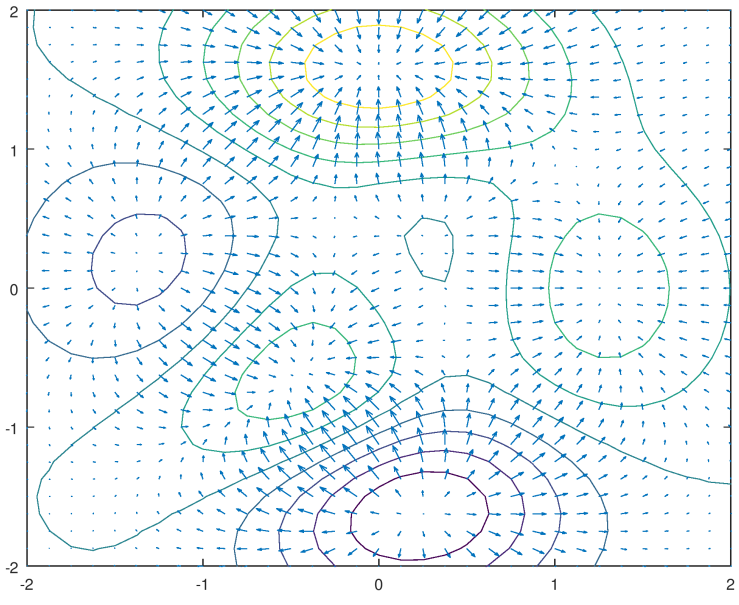


Figure 6.18: Gradient field with contour plot

6.5 Differential equations

6.5.1 Slope fields

The quiver function we used to plot vector fields in Section 6.4 can also be used to plot the slope field of a differential equation. The key is recognizing that a differential equation of the form $dy/dx = f(x, y)$ is a function that gives us slopes, which we can interpret as vectors. This will be illustrated with a simple example.

Example 6.5.1. Plot the slope field along with several solutions of the differential equation

$$\frac{dy}{dx} = x$$

Solution. The solution is $y = \frac{1}{2}x^2 + C$. For differential equations that cannot be solved so easily, plotting the slope field can be used to get a sense of the solutions. In this example, since we know the solution, we can show how the solutions follow the slope field.

We need to define the input range as a meshgrid, define the function, then use the function to calculate slopes. To get a good looking graph, we then scale these slope vectors to a unit length. Finally, we plot some solutions for different values of C.

```
>> % define input values
>> x = linspace(-5, 5, 30);
>> y = x;
>> [X Y] = meshgrid(x, y);

>> % define function
>> f = @(x, y) x;

>> % delta-y, relative to 1 unit delta-x
>> dY = f(X, Y);
>> dX = ones(size(dY));

>> % factor to scale to unit length
>> L = sqrt(1 + dY.^2);

>> % plot the field (note: scale factor 0.5 for shorter arrows)
>> quiver(X, Y, dX./L, dY./L, 0.5)
>> axis([-4 4 -4 4])
>> grid on
>> xlabel('x')
>> ylabel('y')

>> % add some particular solutions to graph for comparison
>> hold on
>> for C = -4:3
     plot(x, 0.5*x.^2 + C, 'r', 'linewidth', 2)
   end
```

The results are shown in Figure 6.19. □

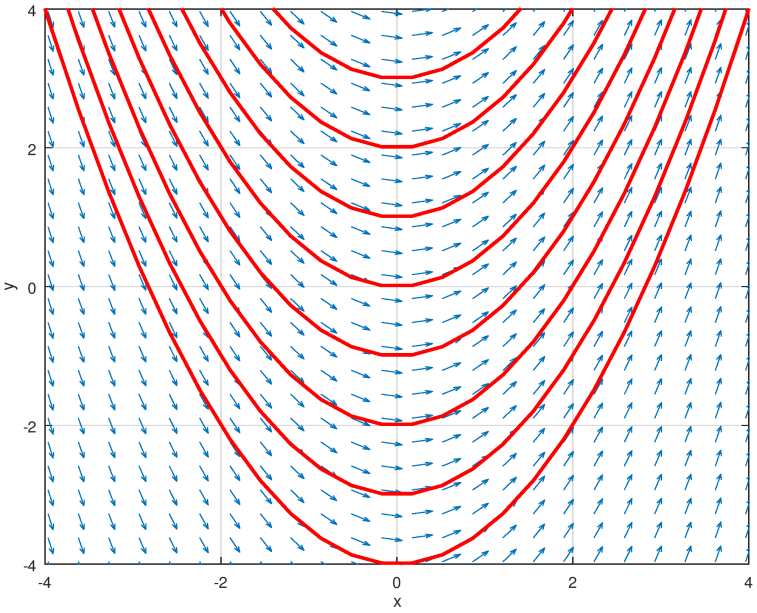


Figure 6.19: Slope field and solutions

6.5.2 Euler's method

Euler's method is probably the simplest numerical technique for solving an ordinary differential equation.

Given a differential equation $y' = f(x, y)$ with initial condition $y(x_0) = y_0$, Euler's method gives approximate solutions:

$$y_{i+1} = y_i + hf(x_i, y_i) \tag{6.7}$$

The value of h is the *step size*. If the interval $[x_0, b]$ is divided into n equally spaced subintervals, then $h = \dfrac{b - x_0}{n}$. To see how this works, let's look at an example.

Example 6.5.2. Solve

$$y' = e^{-3x} - 3y, \quad y(0) = 1$$

on $[0, 3]$ using a step size of 1.

Solution. We will generate a series of approximate y-values at $x = 0, 1, 2, 3$. The value y_0 is given. We compute the remaining values using Equation 6.7. Here is the first step:

$$\begin{aligned} y_1 &= y_0 + hf(x_0, y_0) \\ &= 1 + (1)f(0, 1) \\ &= 1 + (1)(-2) \\ &= -1 \end{aligned}$$

This is then used to compute y_2.

$$\begin{aligned} y_2 &= y_1 + hf(x_1, y_1) \\ &= -1 + (1)f(1, -1) \\ &= 2.0498 \end{aligned}$$

One more step:

$$\begin{aligned} y_3 &= y_2 + hf(x_2, y_2) \\ &= 2.0498 + (1)f(2, 2.0498) \\ &= -4.0971 \end{aligned}$$

Our approximate solutions are summarized in the following table.

x	y
0	1.0000
1	−1.0000
2	2.0498
3	−4.0971

Unfortunately, these solutions are not very accurate. But, we can do much better by decreasing the step size, as shown in the next example. □

These repetitive computations are best implemented in an Octave script. This allows using a smaller step size, which gives a finer range of solution values and also improves the overall accuracy. Refer to [8] for a fuller discussion of the accuracy of Euler's method and a range of more sophisticated algorithms.

Example 6.5.3. Solve

$$y' = e^{-3x} - 3y, \quad y(0) = 1$$

on $[0, 3]$ using a step size of 0.1.

Solution. We will write a fairly general Octave script that can be easily modified for different functions, intervals, and step sizes.

Octave Script 6.4: Euler's method

```
% Euler's method solution for
%   dy/dx = e^(-3x) - 3y, y(0) = 1 on [0, 3]

% define function and initial condition
f = @(x, y) exp(-3*x) - 3*y;
y0 = 1;

% define interval and step size
a = 0;
b = 3;
h = 0.1; % note: step size must divide b-a
n = (b - a)/h;

```

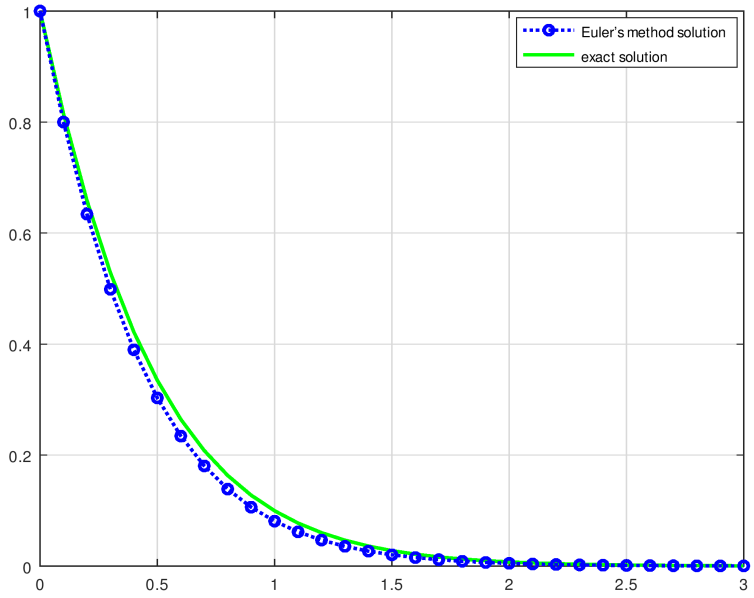


Figure 6.20: Euler's method solution for Example 6.5.3

```
% define x-values
x = [a : h : b];

% calculate y-values
y(1) = y0;
for i = 1:n
    y(i + 1) = y(i) + h*f(x(i), y(i));
end

% plot solutions
>> plot(x, y, 'o:', 'linewidth', 2)
```

Figure 6.20 shows the approximated solution compared to the exact solution, which is known to be $y = e^{-3x}(x + 1)$. □

6.5.3 The Livermore solver

Octave has a built-in function for solving differential equations numerically, called lsode, which implements the Fortran routine of the same name (Livermore solver for ordinary differential equations). The command lsode(f, x_0, t) solves differential equation $dx/dt = f(x, t)$ with initial condition $x(t_0) = x_0$ over the range specified by t. Notice that the initial value x_0 needs to correspond to the first value of the vector **t**. Refer to the documentation for further details.

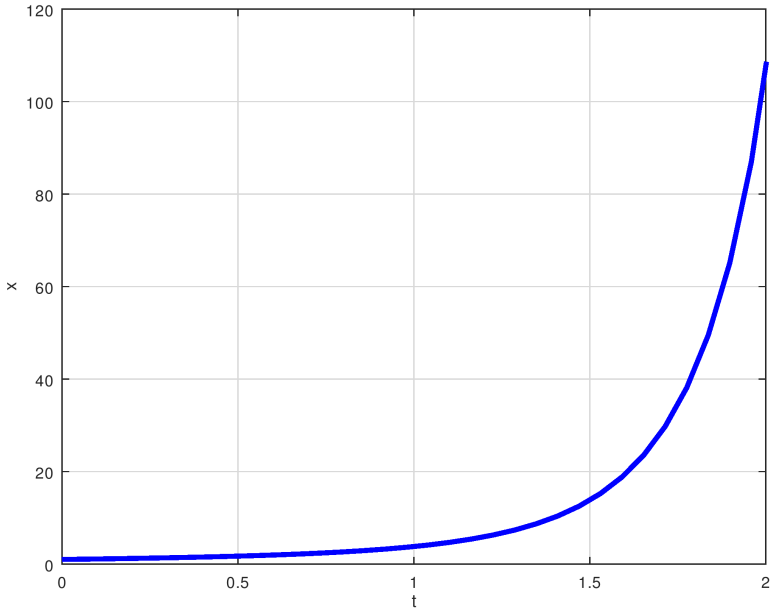


Figure 6.21: lsode numeric solution

Example 6.5.4. Use lsode to solve the differential equation

$$\frac{dx}{dt} = x(t^2 + 1)$$

on $[0, 2]$, with initial condition $x(0) = 1$.

Solution. To solve using lsode, we define the function listing x first, then t.

```
>> % define the function, input values, and initial condition
>> f = @(x, t) x.*(t.^2 + 1);  % x first, then t
>> t = linspace(0, 2, 50);
>> x0 = 1;

>> % calculate the solutions
>> x_sol = lsode(f, x0, t);

>> % plot the solutions
>> plot(t, x_sol, 'linewidth', 2)
>> grid on
>> xlabel('t')
>> ylabel('x')
```

The solution is shown in Figure 6.21. □

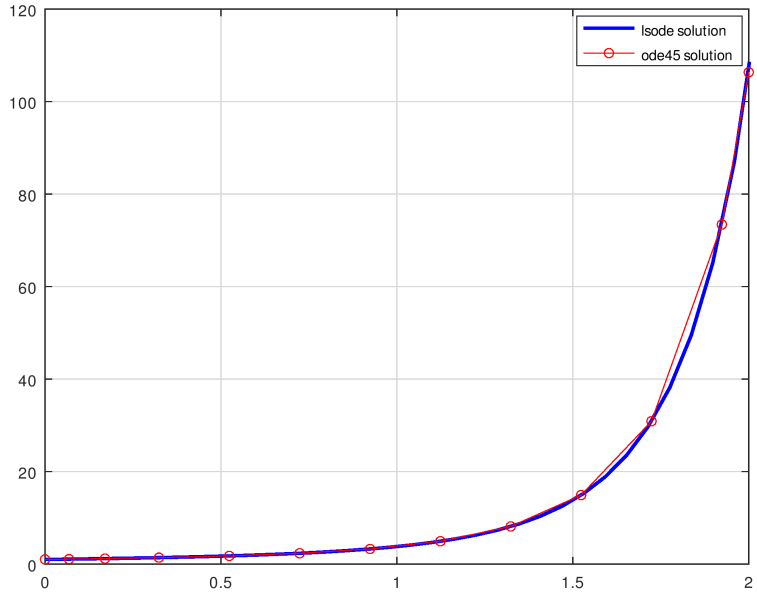


Figure 6.22: ode45 vs lsode numeric solutions

6.5.4 ODE45

For compatibility with MATLAB, several other solvers are available. Let's revisit Example 6.5.4 and solve using the MATLAB-equivalent command ode45.

Example 6.5.5. Use ode45 to solve the differential equation

$$\frac{dx}{dt} = x(t^2 + 1)$$

on $[0, 2]$, with initial condition $x(0) = 1$. Compare to the lsode solution from Example 6.5.4.

Solution. We need to redefine the function. MATLAB convention requires giving the independent variable first, the opposite of what lsode required.

```
>> % define the function, input values, and initial condition
>> f = @(t, x) x.*(t.^2 + 1);    % t first, then x
>> tspan = [0 2];
>> x0 = 1;

>> % calculate the solutions
>> [t_sol, x_sol] = ode45(f, tspan, x0);

>> % plot the solutions
>> plot(t_sol, x_sol, 'o-')
```

Figure 6.22 compares the lsode and ode45 solutions. The solutions seem to agree. □

6.5.5 Exact solutions

The dsolve function, part of the symbolic package, provides a method for finding exact solutions to differential equations. To demonstrate the syntax, we revisit the equation from Example 6.5.3.

Example 6.5.6. Find the general solution for $y' = e^{-3x} - 3y$. Then, find the particular solution if $y(0) = 1$.

Solution. First, the symbolic package must be installed and loaded. Refer to Section 3.4 for details.

To set things up, we declare y as a symbolic function of x.

```
>> pkg load symbolic
>> syms y(x)
```

Now, define the differential equation. We do this using the equality operator, "==".

```
>> ode = diff(y, x)==exp(-3*x) - 3*y
ode = (sym)

  d                        -3*x
  --(y(x)) = -3*y(x) + e
  dx
```

The general solution can now be determined:

```
>> dsolve(ode)
ans = (sym)

                      -3*x
  y(x) = (C1 + x)*e
```

To find the particular solution, we simply need to provide the initial condition.

```
>> dsolve(ode, y(0)==1)
ans = (sym)

                     -3*x
  y(x) = (x + 1)*e
```

This is in good agreement with the numeric solutions determined earlier with Euler's method.

□

Chapter 6 Exercises

1. Let $\mathbf{r}(t) = \langle e^{-t}, t, \sin(t) \rangle$.
   (a) Graph the function on the interval $[0, 2\pi]$.
   (b) Graph the function over the interval $[0, 6\pi]$ and explain the behavior of the graph in the limit as $t \to \infty$. The command comet3(x, y, z) may be helpful.

2. Let $\mathbf{r}(t) = \langle t\cos(t), t\sin(t), t^2 \rangle$.
   (a) Graph the function for $t \in [0, 2\pi]$.
   (b) Find the equation of the line tangent to the curve at $(-\pi, 0, \pi^2)$. Plot the tangent line on the same axes with the curve.

3. Graph each surface.
   (a) $f(x, y) = x^2 + y^2 - 4x + 2y - 1$
   (b) $f(x, y) = \sin(xy) + 1$
   (c) $f(x, y) = 1/(1 - x^2 - y^2)$
   (d) $f(x, y) = \ln(xy - 1)$

4. Let $f(x, y) = \sin(x) + \cos(y)$.
   (a) Graph the surface on $[-2\pi, 2\pi] \times [-2\pi, 2\pi]$.
   (b) Identify any maximums, minimums, or saddle points on the region $(-\pi, \pi) \times (-\pi, \pi)$.
   (c) Graph a contour plot of the surface on $[-\pi, \pi] \times [-\pi, \pi]$ and plot markers at any critical points identified in part (b).
   (d) Calculate $\nabla f(x, y)$ and add a plot of the gradient field to the contour plot.

5. Find a vector function that represents the curve of intersection of the circular cylinder $x^2 + y^2 = 4$ and the parabolic cylinder $z = x^2$. Graph the two surfaces and the curve of intersection.

   The command [X Y Z] = cylinder([2 2]) can be used to obtain a cylinder of radius 2. The output is a cylinder with height 1, which can be stretched as needed to an arbitrary height. For example, if you need height 4, set Z = 4∗Z.

6. Let $f(x, y) = \dfrac{4}{xy}$. Find the equation of the tangent plane at $(2, 2, 1)$. Graph the surface and tangent plane on the same axes over an appropriate domain.

7. Calculate the volume of the bumpy sphere from Example 6.2.4.

8. A cylindrical drill with radius 1 is used to bore a hole through the center of a sphere of radius 5. Graph the ring shaped solid that remains and find its volume.

9. Use dblquad to evaluate the double integral

$$\iint_D x\cos y \; dA$$

   where D is bounded by $y = 0$, $y = x^2$, $x = 0$, and $x = 2$.

10. Let $f(x, y) = x + 2y^2$, defined on $R = [0, 2] \times [0, 4]$. Write a vectorized script to compute a double Riemann sum using midpoints of each subrectangle as the sample points. Use a partition with $m = n = 1000$. Compare your results to the value computed using dblquad ('f', 0, 2, 0, 4) and to the estimate using upper right hand corner sample points, as in Scripts 6.2 and 6.3.

11. Write a function file to implement the vectorized midpoint rule algorithm from Exercise 10. Your function should take as inputs an anonymous two variable function, limits of integration, and a constant term for the number of subdivisions in the partition (let $m = n$). Name the function `dblMid` and test the code on the integral $\int_{-1}^{1}\int_{0}^{2} e^{x^2}\cos(y)\,dx\,dy$, using various values for m. For example,

```
>> f = @(x, y) exp(-x.^2).*cos(y);
>> m = 100;
>> dblMid(f, 0, 2, -1, 1, m)
ans =  1.4845
```

    Compare to the result using dblquad. How large does m need to be before the midpoint algorithm matches Octave's quadrature algorithm, to five significant figures?

12. Let $\mathbf{F}(x, y) = \langle -y, x\rangle$.

    (a) Plot the field on $[-2\pi, 2\pi] \times [-2\pi, 2\pi]$.

    (b) Try to determine visually if the curl of the field is positive, negative, or zero. Confirm your answer using the command curl(X, Y, U, V). This determines the curl at each point in the meshgrid used for the quiver plot. Or load the symbolic package and use the following syntax:

```
>> syms x y z
>> F = [-y x 0]
>> curl(F, {x, y, z})
```

    (c) Try to determine visually if the divergence of the field is positive, negative, or zero. Confirm your answer numerically using the command divergence(X, Y, U, V), or symbolically using divergence(F, {x, y, z}), with F defined as above.

    (d) Calculate the curl and divergence by hand and compare to the numeric results from Octave.

13. Let $\mathbf{F}(x, y) = \tan^{-1}\left(\frac{y}{x}\right)\mathbf{i} + \ln(x^2)\mathbf{j}$.

    (a) Plot the field on $[-2\pi, 2\pi] \times [-2\pi, 2\pi]$.

    (b) Determine visually if the curl is positive or negative. (Hint: the sign of the curl is different for positive and negative x-values.)

    (c) Confirm your answer to part (b) by calculating the curl of the field.

14. Let $\mathbf{F}(x, y, z) = \dfrac{-x}{(x^2+y^2+z^2)^{3/2}}\mathbf{i} + \dfrac{-y}{(x^2+y^2+z^2)^{3/2}}\mathbf{j} + \dfrac{-z}{(x^2+y^2+z^2)^{3/2}}\mathbf{k}$.

    (a) Plot the vector field.

    (b) Show that the field is conservative.

15. Let $\dfrac{dx}{dt} = x(t^2 + 1)$ with initial condition $x(0) = 1$.

    (a) Use Euler's method to solve the differential equation on $[0, 2]$ using a step size $h = 0.1$. Compare to the lsode and ode45 solutions shown in Figure 6.22.

    (b) Use dsolve to find the exact solution.

16. Consider the logistic equation $\dfrac{dy}{dx} = y(1 - y)$.

    (a) Graph the slope field for this equation on $[-6, 6]$.

    (b) Use lsode or ode45 to solve the equation if $y(0) = 0.5$. Graph the solution over the slope field.

    (c) Use dsolve to find first the general solution, then the particular solution given $y(0) = 0.5$.

Chapter 7

Applied projects

What is any of this stuff good for? Lots! This chapter contains several extended projects suitable for calculus and linear algebra students. Major mathematical topics include the singular value decomposition, nonlinear curve-fitting, cubic splines, triangulation, arc length and curvature, space curves and surfaces, and systems of differential equations.

Some projects require packages from Octave Forge. To see a list of your installed packages, type pkg list . If you do not already see the package you need listed, type pkg install −forge NAME. Once installed, the package is loaded with the command: pkg load NAME.

7.1 Digital image compression

How do we reduce large images to manageable file sizes? One approach uses the singular value decomposition (SVD). A digital image can be represented as a matrix, where each entry represents a pixel and we assign a numeric value to each color. The singular values of the matrix are the key. Typically some of these are large, but many are very small. By keeping only the significant singular values and throwing out the rest, we can significantly reduce the amount data that we need to store.

To illustrate the idea, we have imported a small grayscale image file. It is 133×150, which means the matrix has $19,950$ entries. The SVD is then used to generate several approximations at significantly reduced file sizes. The original and reduced images are shown in Figure 7.1 (the original, exact image is at the far right).

The first approximation uses only three singular values. That means we keep three σs, plus three columns of U and three columns of V. We can simply set the other values to 0 (then we don't need to store that data), or delete them. We then multiply these smaller matrices back together to obtain a full size approximation of the original image. The upshot is we only have to store $3 + (3 \times 133) + (3 \times 150) = 852$ data values, compared to the original total of $19,950$. That is only 4% of the original size! Keeping a few more singular values, the second approximation uses 10 singular values and is 14% of the original size. The image quality is not bad, considering how much of the original data we threw away. The third approximation, with 30 singular values,

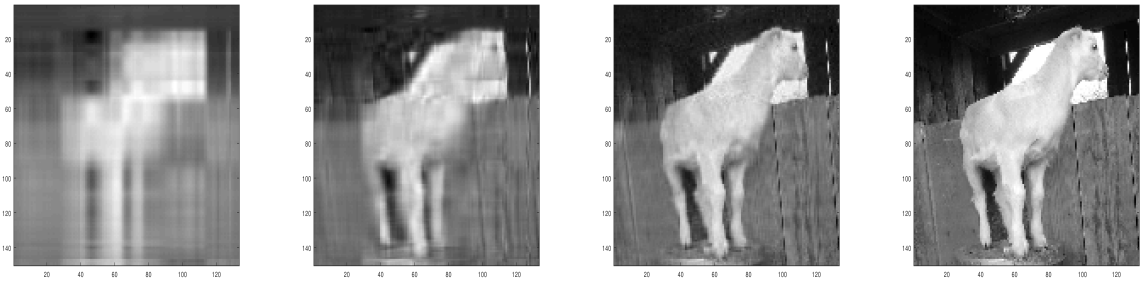


Figure 7.1: SVD approximations

looks almost as good as the original. But, it is only 43% of the original size. For comparison, the exact image is shown on the far right.

Octave supports several image file types. We will use JPG files, which are loaded as RGB (red, green, blue) images, represented as a set of three $m \times n$ matrices containing the color values for each pixel. For simplicity, we will convert this to a single $m \times n$ grayscale matrix. Some of the Octave commands needed for basic image processing are listed in the table below.

IMAGE PROCESSING COMMANDS

Syntax	Description
pkg install −forge image ...	install the image package from Octave Forge
pkg load image	load the image package
im = imread('filename.jpg');	load an image
name = imresize(im, 0.5); ..	reduce image size by a specified factor (e.g., 0.5)
name = rgb2gray(im);	convert to grayscale
imshow(im)	display an image
imagesc(im)	display a matrix as a scaled image
colormap('gray')	set colormap to grayscale
colormap('default')	restore colormap to default

Problems

1. For this problem, you will use the SVD to produce a compressed image using k singular values. You choose k (something between 5 and 50 would be suitable). To begin, you will need a digital photo in JPG format. Make sure you have loaded the image package.

   (a) Load the image in Octave as "imcolor," then convert to a grayscale image.

   (b) Check the size of your grayscale image and if it is larger than approximately 320×280, determine an appropriate reduction factor and reduce it. Reducing to a modest size makes it easier to open the variables in the variable editor to inspect their values.

   Name the reduced, grayscale format image "im." This is the image that will be compressed via the SVD method. Display the reduced grayscale image using imagesc and verify that it still looks like the original. Include a copy of this grayscale image with your problem solutions and state its size.

(c) Find the SVD of the matrix representation "im."

(d) Use the SVD to calculate an approximation using k singular values. That means you should only keep the first k columns of U, the k largest values of Σ, and the first k columns of V. Set the other values to 0 (or delete the extra columns altogether), then compute $U\Sigma V^T$ to recover an approximation of the original image. Save it as "im2" and display it using imagesc. Include a copy of the reduced image with your problem solution.

(e) How many nonzero values are saved in the compressed factorization compared to the original?

2. Using the "outer product" expansion of $A = U\Sigma V^T$, the matrix A can be calculated column-by-column as

$$A = \sigma_1 \mathbf{u}_1 \mathbf{v}_1^T + \sigma_2 \mathbf{u}_2 \mathbf{v}_2^T + \cdots + \sigma_r \mathbf{u}_r \mathbf{v}_r^T$$

where each term is an $m \times n$ matrix. For any $k < r$, the sum of the first k terms will be an approximation using $km + kn + k$ data values.

(a) Use the steps outlined in problem 1 to load a grayscale image matrix.

(b) Write a for-loop to generate an outer product expansion using k singular values.

(c) Run the loop through 1, 5, 10, 20, and 30 iterations, saving a copy of the output image each time.

(d) How many singular values do you think are needed before the quality of the reduced image is "good enough"?

(e) At what k-value does the SVD approximation actually require saving *more* data than the original image?

3. Use the iterative outer product method of problem 2 to create a slide show showing the progressive quality improvement as the number of singular values increases from 1 up to the point where the approximation and original are indistinguishable.

You will need to use a loop that produces an approximation for each i from 1 to k. Some special formatting is needed to create a series of file names that increment as you cycle through the loop. Load your grayscale image and find the SVD. Then use the code below as a template to generate a series of progressively better approximations.

```
>> % initialize approximation and set number of singular values
>> approx = zeros(size(im));
>> k = 25

>> % loop to create approximations and save as image files
>> for i = 1:k
>>  approx = approx + S(i, i)*U(:, i)*V(:, i)';
>>  h = imagesc(approx);
>>  name = sprintf('%s%d.png', 'approx', i);
>>  saveas(h, name)
>> end
```

7.2 The Gini index

The subject of wealth and income inequality has featured prominently in the media and political campaigns recently. Is there a fair, mathematically quantifiable method to describe income inequality and track how it changes over time? This project will look at how Lorenz curves and the Gini index are used to measure inequality and make comparisons.

The US Census Bureau reports shares of aggregate income by *quintile*. Quintiles divide the population into equal fifths. For example, in 2018, the bottom quintile or poorest 20% of the population earned 3.1% of the total income. In a perfectly egalitarian society (equal income distribution), each 20% of the population would earn 20% of the income. The table below shows the share of the total income at various income levels as reported by US Census Bureau for 2018[1].

INCOME DISTRIBUTION

Shares of Aggregate Income	Mean Income	Share of Income	Cumulative Share
Lowest Quintile	$13,775	3.1%	3.1%
Second Quintile	$37,293	8.3%	
Third Quintile	$65,572	14.1%	
Fourth Quintile	$101,570	22.6%	
Highest Quintile	$233,895	52.0%	100%

Problems

1. Fill in the column for cumulative share of income in the table above. This gives the share of total income earned by the bottom 20%, the bottom 40%, the bottom 60%, the bottom 80%, and finally by 100%.

2. A *Lorenz curve* is obtained by plotting $(a/100, b/100)$ if the bottom a% earn at most b% of the income. Fill in the table below with the decimal form of the cumulative shares, then plot the points and sketch in a smooth curve.

x	$L(x)$
0.00	0.000
0.20	
0.40	
0.60	
0.80	
1.00	1.000

   Notice that the curve passes through $(0, 0)$ and $(1, 1)$. The curve is also increasing and concave up (it never crosses above the line $y = x$ or drops below the line $y = 0$). It lies in between the curve of perfect equality, $y = x$, and perfect inequality. Maximum inequality would be when the top income earner earns 100% of the income and everyone else earns 0. This is the line $y = 0$ for $0 < x < 1$, then a jump to the point $(1, 1)$. Plot the curves of perfect equality and inequality.

[1]Source: *Income and Poverty in the United States: 2018*, US Census Bureau (September, 2019). Retrieved from https://www.census.gov/content/dam/Census/library/publications/2019/demo/p60-266.pdf.

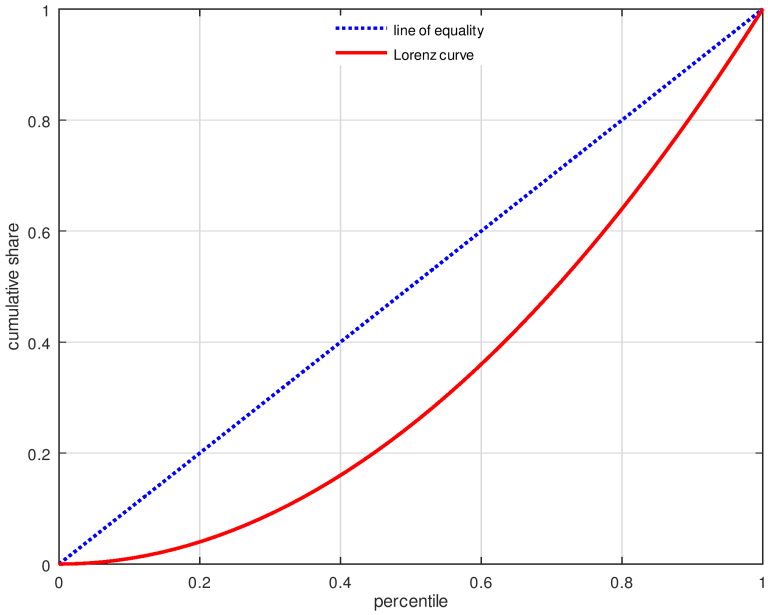


Figure 7.2: Lorenz curves

3. Lorenz functions satisfy these four properties:

$$\begin{array}{llll} \text{(i)} & L(0) = 0 & \text{(iii)} & L'(x) > 0 \text{ for } 0 < x < 1 \\ \text{(ii)} & L(1) = 1 & \text{(iv)} & L''(x) > 0 \text{ for } 0 < x < 1 \end{array}$$

Show that each of the following functions are Lorenz functions by verifying properties i–iv.

(a) $L(x) = x^2$

(b) $L(x) = x^n$, where $n \geq 2$

(c) $L(x) = a(e^x - 1)$, where $a = 1/(e - 1)$

(d) $L(x) = a(e^{kx} - 1)$, where $a = 1/(e^k - 1)$, $k > 0$

4. The *Gini index* (or *Gini coefficient*), G, measures how much a given income distribution differs from perfect equality. Specifically, it is defined to be the area between the line of perfect equality and the Lorenz curve, divided by the total area under the line of perfect equality. Show that

$$G = 2 \int_0^1 [x - L(x)] \; dx$$

What are the values of G for perfect equality and for maximum inequality?

5. Load the optimization package. Then use the data from problem 2 and the nonlinear curve-fitting function nonlin_curvefit to approximate the Lorenz curve for 2018 U.S. income as a function of the form $L(x) = x^n$. Use this model to calculate the Gini index. The US Census Bureau reports the Gini index as 0.486. How well does your answer agree?

The following code shows the basic syntax, assuming xdata and ydata contain the x- and $L(x)$-values from the income distribution.

```
>> % load the optim package (must already be installed)
>> pkg load optim

>> % define the model
>> % power function y = x^n
>> f = @(n, x) x.^n;

>> % initial guess for parameter n
>> init = 2;

>> % calculate the model
>> [n, model_values] = nonlin_curvefit(f, init, xdata, ydata)
```

Plot the data values and the Lorenz function together on the same axes. Include a legend and axis labels.

6. Repeat problem 5 using a Lorenz function of the form $L(x) = a(e^{kx} - 1)$, where $a = 1/(e^k - 1)$. How does the value of G compare to the result from problem 5 and the US Census Bureau figure?

7. How has income inequality in the U.S. changed in past 50 years? The following table gives values of the Lorenz function for the U.S. every ten years starting in 1970 (derived from US Census Bureau figures). Fill in the final column with either the 2018 data listed above or the most recent data available on the US Census Bureau website.

x	1970	1980	1990	2000	2010	current data
0.00	0.000	0.000	0.000	0.000	0.000	0.000
0.20	0.041	0.042	0.038	0.036	0.033	
0.40	0.149	0.144	0.134	0.125	0.118	
0.60	0.323	0.312	0.293	0.273	0.264	
0.80	0.568	0.559	0.533	0.503	0.498	
1.00	1.000	1.000	1.000	1.000	1.000	1.000

Plot the Lorenz functions together on the same set of axes. Is a trend evident? Use the power function method of problem 5 to calculate the Gini index for each of those years. How has the Gini index changed since 1970? If it has gone up, by what percent has it increased from 1970 to the present?

8. How has income inequality in the U.S. changed in the last century? Do some research and cite reputable sources to describe how the Gini index has changed over the last 100 years.

9. How does income inequality in the U.S. compare to the rest of the world? Again, do some basic research and cite reputable sources to back up your assessment of how the U.S. Gini index compares to other nations.

7.3 Designing a helical strake

In this project, we will tackle a real engineering problem. First, you will calculate the dimensions necessary to build a "helical strake" and construct a computer model to visualize it. Then you will build a physical model to test whether our methods work in the real world. A strake is a metal strip, used to reduce vibrations due to wind shear, attached on edge to the outside of a smokestack or other cylinder in a spiral. See Figure 7.3[2].

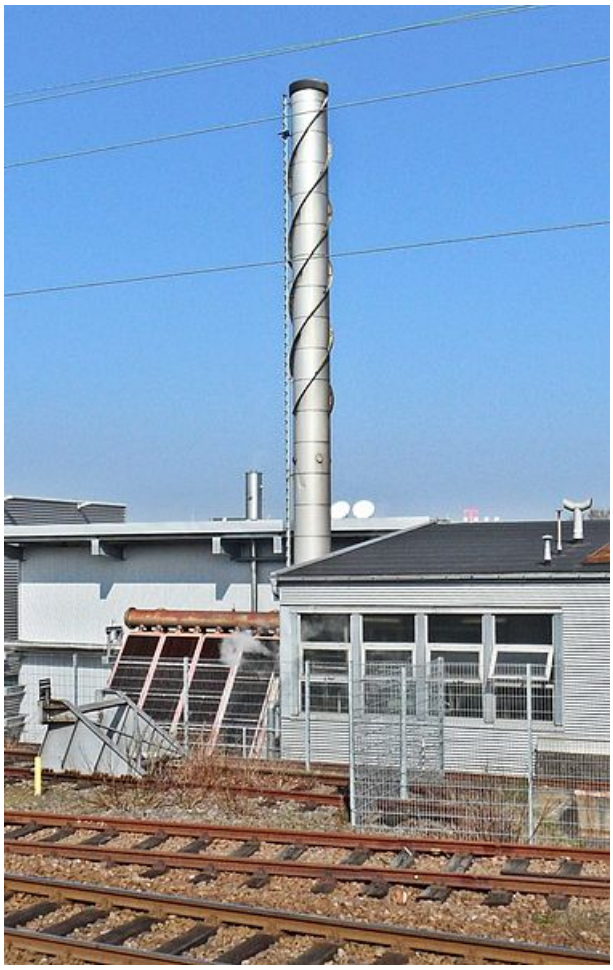


Figure 7.3: Chimney with helical strake

There are two basic mathematical problems that must be solved to construct this. First, we need to know the (inside) linear length of the metal strip – that is the arc length of the helix:

$$s = \int_a^b \left\| \mathbf{r}'(t) \right\| \, dt$$

Secondly, the circular metal strips are cut out of flat metal sheets, then twisted into shape. What inside radius will make the strake fit flush against the smokestack? The correct approach

[2] *Steel chimney with spiral,* by "StomBer" (CC-BY). https://commons.wikimedia.org/wiki/File:SchornsteinwendelSKL.jpg

is to build the strake with the right *radius of curvature*[3]. The radius of curvature is $\rho = 1/\kappa$, where κ is the curvature, defined as the rate of change of the unit tangent with respect to arc length:

$$\kappa = \left\| \frac{d\mathbf{T}}{ds} \right\|$$

Refer to [5, §11.5] for easier-to-use computational formulas for curvature.

Problems

Choose a cylinder. You could use something as small as an empty paper towel roll or something as large as a cardboard concrete pier form. Measure the height and the radius of your cylinder. We want the helix to make exactly one revolution over the height of the tube. Before building the actual model, we will construct a virtual model.

1. Determine the equations of your cylinder and the helix that fits flush against it.

2. Plot a graph of the cylinder and helix on the same axes. Decide on a width for your strake and then plot the outside radius on the same axes to complete the model. Here are some commands that will help you plot the cylinder:

   ```
   >> [X Y Z] = cylinder([r r]);
   >> Z = h*Z;
   >> surf(X, Y, Z)
   ```

   This generates the X, Y, and Z meshgrid arrays for a cylinder with radius r and height 1. The values of r are for the radii at the top and the bottom. For an ordinary right-circular cylinder, both numbers are the same. To stretch the height out to match the height of your cylinder, we multiply the Z-coordinate by h. If you don't like the way the plot from the surf command looks, try mesh(X, Y, Z). Once you have the cylinder plotted, use hold on and add a plot of the helix, with radius r. That represents the inside of the strake. To plot the outside edge, plot another helix of radius $r + w$, where w is the width of the strake. Optionally, add two line segments to join the two helixes at the ends.

3. Calculate the arc length of the helix.

4. Calculate the radius of curvature of the helix.

5. Build a physical model. You can use construction paper, poster board, cardboard, or foam board. The material needs to be flexible enough to be twisted into the correct shape. To attach the spiral to your cylinder, you can use tape or glue. Duct tape or packaging tape should work, but a stout glue will show the interface between the strake and cylinder more clearly. In any case, we don't need something attractive; we just need to know whether the pieces fit.

   Choose an appropriate outside radius for the strake, then cut out the pieces and build the model. You may need to use a nail or thumb tack and a measured length of string to mark your radius before cutting.

[3] Morgan, Frank. *Riemannian Geometry: A Beginner's Guide* (Second Edition). AK Peters, Ltd., Wellesley, MA (1998).

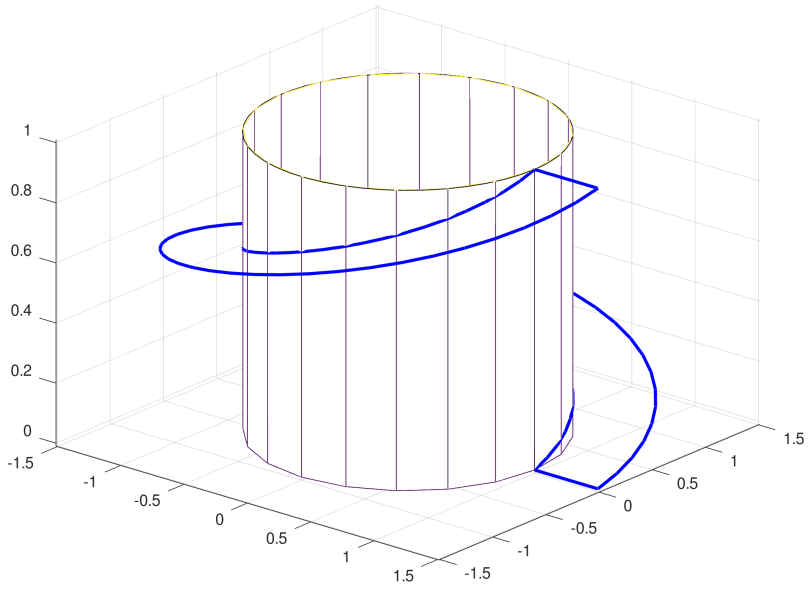


Figure 7.4: Computer model

6. Do the calculations agree with the reality of the physical model?

   (a) Does the inside length of strake, as measured with a string, equal the calculated arc length?

   (b) Do the pieces fit flush against the cylinder?

   (c) Does the helix make exactly one revolution?

   If something did not work out correctly, try to explain what you think went wrong.

7. Many chimneys have a wider base and narrow as the height increases. How does the problem change if we wish to wrap our strake around a more complex shape?

   (a) Find the correct equations and create a computer model of a strake wrapped around a frustum of a cone with lower base radius 5, upper radius of 1, and height 10.

   (b) Construct the computer model to illustrate the strake.

   (c) Calculate the arc length of the helix.

   (d) Calculate the curvature of the strake for $t = 0, \pi/2, \pi, 3\pi/2,$ and 2π, where $t \in [0, 2\pi]$ corresponds to one complete revolution.

   (e) You do not need to construct a physical model, but briefly discuss what difficulties would be encountered if we wished to construct this model.

7.4 3D-printing

If $z = f(x, y)$ is nonnegative on D, then $\iint_D f(x, y)\, dA$ represents the volume under the graph of f, above the region D in the xy-plane. We have already seen to how to calculate such volumes and how to visualize the surfaces by using three dimensional graphs in Octave. In this project, we will convert 3D-surfaces into solid blocks which can be printed in three dimensions. In addition to allowing us a more concrete means of modeling a mathematical surface, we can physically measure the volume of the object (by displacement) and compare to the theoretical results as predicted by calculus.

These printers require closed, "water tight" solids, typically produced using a triangular mesh and saved in an STL file format (STereoLithography). There are several MATLAB script files available in the MathWorks File Exchange for producing the figures and STL-output we need. While these scripts may run in Octave, it is a violation of the terms of service to use content from the Mathworks File Exchange in non-Mathworks software. If you are using a licensed version of MATLAB, try:

- http://www.mathworks.com/matlabcentral/fileexchange/30709-surf2solid
  surf2solid takes a function of two variables and turns it into a solid block by adding a "curtain" from the boundary of the surface down to the xy-plane and joining this to a flat plane below
- http://www.mathworks.com/matlabcentral/fileexchange/4512-surf2stl
  surf2stl takes a solid rectangular meshgrid and converts into a triangulated STL-format surface

Inspired by these functions, we will write our own Octave function for this project. We have two goals:

1. For a surface $z = f(x, y)$, create a solid block that represents the volume below the surface and above a rectangular domain.
2. For any surface defined by meshgrid variables X, Y, Z, generate a triangulation and write an STL file.

STeroLithography files in ASCII format have a simple structure:

```
solid name
  facet normal n1 n2 n3
    outer loop
      vertex v1 v2 v3
      vertex v1 v2 v3
      vertex v1 v2 v3
    endloop
  endfacet

endsolid name
```

If we start from a rectangular grid, as we do with a surface generated from meshgrid variables, to produce a triangulation we merely need to split each subrectangle of the partition into two triangles. The STL-format requires the vertices of these triangles, plus a unit normal vector for each of the triangular facets. Despite the requirement for a normal vector in the file standard, it is redundant, as the vertices alone uniquely determine the solid. Thus some software ignores the normal vectors. But, we will attempt to include correct unit normals calculated using a cross product.

In the `mesh2stl.m` function file below, lines 12–29 are all that is needed to triangulate an (X, Y, Z)-surface from Octave. Lines 92–123 will write the output in the standard file format. But, if we pass the function a minimum thickness, delta, then additional code is used. Assuming $z = f(x, y)$ is a function over a rectangular domain, it is relatively simple to produce a solid block by dropping a "curtain" from the edge of the surface down to the xy-plane and closing off the solid with a flat plane for the base. Lines 31–90 handle this.

This function file can be downloaded from https://gist.github.com/jalachniet/.

Octave Script 7.1: mesh2stl function file

```
1  function mesh2stl(filename, X, Y, Z, delta)
2  %MESH2STL writes an .stl (STereoLithography) file from meshgrid variables
3  %  mesh2stl('filename', X, Y, Z)
4  %     produces a triangulated mesh from meshgrid variables X, Y, Z
5  %     and writes output to 'filename' in stl format
6  %  --X, Y, Z must be two dimensional arrays of the same size
7  %  --optional: if delta is provided, produce a solid block from the graph
8  %              of surface z = f(x, y)
9  %              where 'delta' gives minimum thickness between base of block
10 %              and graph of surface
11
12 % determine dimensions
13 m = size(Z, 1);
14 n = size(Z, 2);
15
16 % construct triangular facets for surface
17 k = 0;
18 for i = 1:m-1
19   for j = 1:n-1
20     k = k + 1;
21     F(:, :, k) = [X(i, j),     Y(i, j),     Z(i, j);
22                   X(i, j+1),   Y(i, j+1),   Z(i, j+1);
23                   X(i+1, j+1), Y(i+1, j+1), Z(i+1, j+1)];
24     k = k + 1;
25     F(:, :, k) = [X(i+1, j+1), Y(i+1, j+1), Z(i+1, j+1);
26                   X(i+1, j),   Y(i+1, j),   Z(i+1, j);
27                   X(i, j),     Y(i, j),     Z(i, j)];
28   end
29 end
30
31 if (nargin > 4)
32   % calculate elevation and midpoint of base
33   z_base = min(min(Z)) - delta;
34   x_mid = (min(min(X)) + max(max(X)))/2;
```

```
35  y_mid = (min(min(Y)) + max(max(Y)))/2;
36
37  % construct triangular facets for 'curtain' and base
38  for i = 1:n-1
39    k = k + 1;
40    F(:, :, k) = [X(1, i),   Y(1, i),   Z(1, i);
41                  X(1, i),   Y(1, i),   z_base;
42                  X(1, i+1), Y(1, i+1), Z(1, i+1)];
43    k = k + 1;
44    F(:, :, k) = [X(1, i+1), Y(1, i+1), Z(1, i+1);
45                  X(1, i),   Y(1, i),   z_base;
46                  X(1, i+1), Y(1, i+1), z_base];
47    k = k + 1;
48    F(:, :, k) = [x_mid,     y_mid,     z_base;
49                  X(1, i),   Y(1, i),   z_base;
50                  X(1, i+1), Y(1, i+1), z_base];
51    k = k + 1;
52    F(:, :, k) = [X(m, i),   Y(m, i),   Z(1, i);
53                  X(m, i),   Y(m, i),   z_base;
54                  X(m, i+1), Y(m, i+1), Z(1, i+1)];
55    k = k + 1;
56    F(:, :, k) = [X(m, i+1), Y(m, i+1), Z(1, i+1);
57                  X(m, i),   Y(m, i),   z_base;
58                  X(m, i+1), Y(m, i+1), z_base];
59    k = k + 1;
60    F(:, :, k) = [x_mid,     y_mid,     z_base;
61                  X(m, i),   Y(m, i),   z_base;
62                  X(m, i+1), Y(m, i+1), z_base];
63  end
64
65  for j = 1:m-1
66    k = k + 1;
67    F(:, :, k) = [X(j, 1),   Y(j, 1),   Z(j, 1);
68                  X(j, 1),   Y(j, 1),   z_base;
69                  X(j+1, 1), Y(j+1, i), Z(j+1, 1)];
70    k = k + 1;
71    F(:, :, k) = [X(j+1, 1), Y(j+1, 1), Z(j+1, 1);
72                  X(j, 1),   Y(j, 1),   z_base;
73                  X(j+1, 1), Y(j+1, 1), z_base];
74    k = k + 1;
75    F(:, :, k) = [x_mid,     y_mid,     z_base;
76                  X(j, 1),   Y(j, 1),   z_base;
77                  X(j+1, 1), Y(j+1, 1), z_base];
78    k = k + 1;
79    F(:, :, k) = [X(j, n),   Y(j, n),   Z(j, n);
80                  X(j, n),   Y(j, n),   z_base;
81                  X(j+1, n), Y(j+1, n), Z(j+1, n)];
82    k = k + 1;
83    F(:, :, k) = [X(j+1, n), Y(j+1, n), Z(j+1, n);
84                  X(j, n),   Y(j, n),   z_base;
85                  X(j+1, n), Y(j+1, n), z_base];
86    k = k + 1;
87    F(:, :, k) = [x_mid,     y_mid,     z_base;
88                  X(j, n),   Y(j, n),   z_base;
```

```
                    X(j+1, n), Y(j+1, n), z_base];
  end
end

% number of triangular facets
num_facets = k;

% save in stl format
fid = fopen(filename, 'w');
title_str = sprintf('Created with GNU Octave %s', datestr(now));
fprintf(fid,'solid %s\r\n', title_str);
for k = 1:num_facets
   % vertices
   p1 = [F(1, 1, k) F(1, 2, k) F(1, 3, k)];
   p2 = [F(2, 1, k) F(2, 2, k) F(2, 3, k)];
   p3 = [F(3, 1, k) F(3, 2, k) F(3, 3, k)];

   % normal vector
   if ((p1 ~= p2) & (p1 ~= p3) & (p2 ~= p3))
     n = cross(p2-p1, p3-p1)./norm(cross(p2-p1, p3-p1));
   else
     n = [0 0 0]; % unable to calculate normal vector
   end

   % write facets
   fprintf(fid, 'facet normal %.7E %.7E %.7E\r\n', n(1), n(2), n(3));
   fprintf(fid, 'outer loop\r\n');
   fprintf(fid, 'vertex %.7E %.7E %.7E\r\n', p1);
   fprintf(fid, 'vertex %.7E %.7E %.7E\r\n', p2);
   fprintf(fid, 'vertex %.7E %.7E %.7E\r\n', p3);
   fprintf(fid, 'endloop\r\n');
   fprintf(fid, 'endfacet\r\n');
end

fprintf(fid, 'endsolid %s\r\n', title_str);
fclose(fid);

disp('Number of triangular facets:'), disp(num_facets)
```

Let's try producing a solid printable block with a modified version of the "sombrero" function,

$$f(x,y) = 10\left(\frac{\sin(\sqrt{x^2+y^2})}{\sqrt{x^2+y^2}}\right) + 3$$

```
>> [X Y Z] = sombrero;
>> Z = 10*Z + 3;
>> surf(X, Y, Z);
```

Now, we use `mesh2stl` to produce a solid figure and STL file.

```
>> mesh2stl('sombrero.stl', X, Y, Z, 2);
Number of triangular facets:
 3680
```

The output file is readable by most 3D-printer software. The option delta = 2 tells the function to attempt to produce a solid block with a minimum thickness of 2. Note that units in STL files are arbitrary, but most 3D printer software will interpret 1 unit in Octave as 1 mm. Thus our figure has a minimum thickness of 2 mm, measured between the base and the lowest point on the surface.

Use your 3D printer software or an online STL viewer to preview the results and check the size in millimeters. Our figure is quite small, only $16 \times 16 \times 14$ mm. It is generally an easy matter to resize to the desired final dimensions using printer software. Alternately, you can manually scale X, Y, and Z before writing the file.

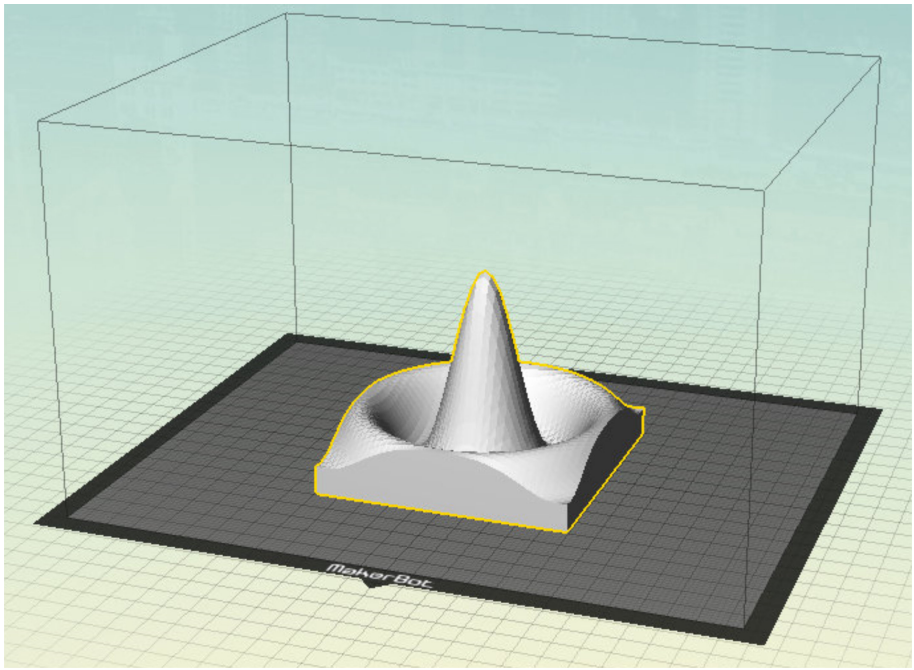


Figure 7.5: STL file ready for printing

Problems

1. Use integration to determine the volume of the modified sombrero function generated above. You will need to take into account the difference in elevation between the base of the block and the lowest point on the surface.

2. Consider the example Octave surface "peaks."

   ```
   >> [X, Y, Z] = peaks;
   ```

   Scale Z by a factor of 1/5 and shift up 2 units.

   ```
   >> Z = Z/5 + 2;
   ```

   Now convert the X, Y, Z-meshgrid variables to a printable STL-format solid block. Calculate the volume of the resulting solid figure.

3. Let $f(x, y) = \sin(xy) + 1$.

   (a) Compute (by hand) the volume under the graph of f over the region $[-\pi, \pi] \times [-\pi, \pi]$.

   (b) Graph the function over the appropriate domain.

   (c) Convert the resulting [X, Y, Z] meshgrid variables to a solid block, saved in STL-format.

4. Consider the "bumpy sphere" from Example 6.2.4,

$$\rho = 1 + \frac{1}{4}\sin(5\phi)\cos(6\theta)$$

   where $0 \leq \theta \leq 2\pi, 0 \leq \phi \leq \pi$.

   (a) Set-up a triple integral in spherical coordinates to find the volume of this solid.

   (b) Evaluate the integral to find a decimal approximation for the volume. You may use technology. Complete the first trivial integration by hand. Then define $\rho = f(\theta, \phi)$ and use dblquad command on the remaining double integral.

   (c) Use the symbolic package to find the exact volume of the bumpy sphere. Compare to the answer from part (b).

   (d) Graph the solid. You will need to create a $\theta\phi$-meshgrid, then calculate ρ using these meshgrid variables. Finally, X, Y, and Z are calculated using the standard spherical to rectangular coordinate identities. To scale the size up, multiply each X, Y, and Z matrix by a factor of 50.

   (e) Convert to STL-format. Note that this figure is already a closed solid, so do not use the delta option.

5. Suppose we have a solid radius-3 hemisphere and drill a radius-2 cylinder through the center. What volume remains? In rectangular coordinates, this is the volume under $z = \sqrt{9 - x^2 - y^2}$ and above a washer shaped region in the xy-plane with inner radius 2 and outer radius 3.

   (a) Sketch the region and calculate the volume by hand. Use polar coordinates.

   (b) Use an $r\theta$-polar meshgrid to produce a 3D graph.

   (c) As written, our mesh2stl function does not work to produce a solid block for a surface defined in polar coordinates. So, instead, use the following code to produce a version of the solid in rectangular coordinates:

```
>> % define f to be 0 outside of relevant washer shaped region
>> f = @(x, y) sqrt(9 - x.^2 - y.^2) .* ((x.^2 + y.^2 < 9) & (x
   .^2 + y.^2 > 4));

>> x = linspace(-3, 3, 100);  % define the domain
>> y = x;
>> [X Y] = meshgrid(x, y);

>> Z = f(X, Y);  % evaluate the function and plot graph
>> surf(X, Y, Z)

>> % convert to .stl format
>> mesh2stl('ring2.stl', 50*X, 50*Y, 50*Z, 0)
```

6. 3D printing can also help with visualizing space curves. However, a curve, on its own, has no thickness and therefore cannot be printed. A typical approach is to replace the curve with a thin tube. Script files are available to do this, for example `tubeplot.m` available from http://www.aleph.se/Nada/Ray/Tubeplot/tubeplot.html.

   Even if you do not plan to 3D-print, a tubeplot can help with visualizing some complex space curves.

   (a) Let $\mathbf{r}(t) = \cos(t)\mathbf{i} + \sin(t)\mathbf{j} + 0.3\sin(2t)\mathbf{k}$. Graph the curve over the interval $[0, 2\pi)$. This is called the "Roller Coaster Curve" ([7, §3.1 Exercise 37]).

   (b) Create a 0.1-radius tube plot of the curve. Here is the correct syntax for the tubeplot script:

   ```
   >> [X Y Z] = tubeplot(x, y, z, 0.1);
   ```

   (c) View the plot using surf(X, Y, Z), then save in STL-format. The figure is essentially already a closed solid, so no option is needed for delta.

7. Here is your chance to be creative! Create a mathematically defined 3D-solid of your own design. You can use a rectangular function of two variables, or you may wish to try something in polar, cylindrical, or spherical form. In the end, it must be a closed solid that you can convert to a printer-ready STL-format. You may get ideas from your calculus book, experimentation in Octave, and/or online image searches.

   (a) Define your function and state its domain (over which you will print the object).

   (b) Graph it.

   (c) Scale to a modest size (base roughly 50×50 mm) and convert to STL.

   (d) Give a description of any interesting mathematical properties of the object and include any relevant computations. Examples:

   - The volume calculated by integration
   - Location of maximums, minimums, or saddle points
   - A point where the limiting value is different along different lines of approach
   - A discontinuity

8. After one of your solid objects from problems 1–4 is printed, take it to the lab and measure its volume directly by fluid displacement. How does the measured volume compare to the calculated volume? Be sure to take into account any scaling factor used for the printed version. Give the absolute and relative volume errors and comment on any discrepancies.

7.5 Modeling a cave passage

In a cave survey, three dimensional data is collected between survey points. There are three measurements: distance, compass bearing (0° to 360° azimuth), and inclination (−90° to 90° vertical angle). The first step in producing a cave map is to reduce this data to rectangular coordinates and generate line plots of the survey. We will then construct mathematical models for the walls, floor and ceiling, and use these to set-up integrals to estimate the floor area, average passage height, and overall volume of the cave. This information might be useful, for example, to a scientist who wished to determine the amount of limestone that dissolved during the formation of the cave, or the size of the stream which once flowed through it.

Survey data for *Skunk Cave* (Smyth County, Virginia) was collected during fieldwork by members of the Walker Mountain Grotto caving club and the Wytheville Community College Outdoor Club. The raw data is given below.

Distance (ft)	Azimuth (°)	Inclination (°)
30.05	248.5	−15.5
10.30	237.5	−25.5
3.20	245.0	11.0
17.00	269.0	−5.0
10.00	271.0	−10.0
14.35	280.0	3.0
11.50	308.5	6.0
49.65	296.0	12.5
5.30	315.0	−23.0

Converting this spherical data to rectangular coordinates is accomplished with the following transformations:

$$\begin{cases} x &= d\cos(\phi)\sin(\theta) \\ y &= d\cos(\phi)\cos(\theta) \\ z &= d\sin(\phi) \end{cases}$$

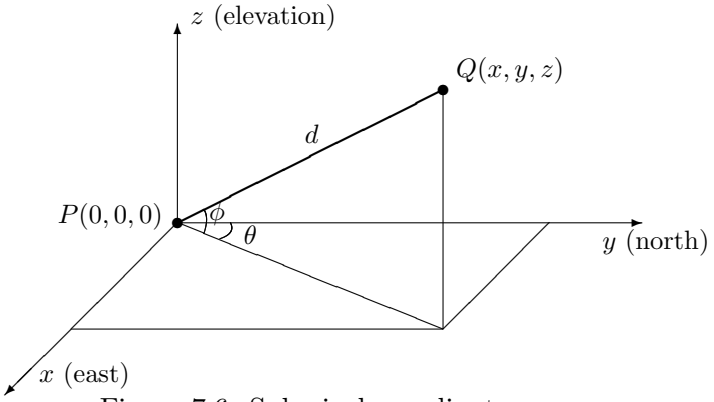


Figure 7.6: Spherical coordinates

Note that these formulas are slightly different from the standard spherical to rectangular identities, to reflect how the measurements are actually taken in the field. To apply these formulas, we must use an iterative process. Start from the point $(0, 0, 0)$. The (x, y, z)-coordinates of the next point are found using the given formulas. Then, this becomes the starting point for the next transformation. Thus, for each $i = 2, 3, 4, \ldots, 10$, the relationship is:

$$\begin{aligned} x_i &= x_{i-1} + d_{i-1}\cos(\phi_{i-1})\sin(\theta_{i-1}) \\ y_i &= y_{i-1} + d_{i-1}\cos(\phi_{i-1})\cos(\theta_{i-1}) \\ z_i &= z_{i-1} + d_{i-1}\sin(\phi_{i-1}) \end{aligned}$$

Part 1: The survey line

1. Determine rectangular coordinates for each point in the survey. These should be in the form of 10×1 column vectors for x, y, and z. You can carry out each step one a time, but using a loop in Octave will be more efficient. Note that you must convert the degree angle measures to radians to obtain correct results.

2. Plot the overhead plan view using the command `plot(x, y)`. For a more accurate perspective, you can force equal x- and y-axis scales by using the command `axis('equal')`.

3. Notice that the passage is aligned primarily in an east-west direction. Plot an east-west profile view by using the command `plot(x, z)`. Force equal axes for a more accurate view.

4. Plot a 3-dimensional model using the command `plot3(x, y, z)`.

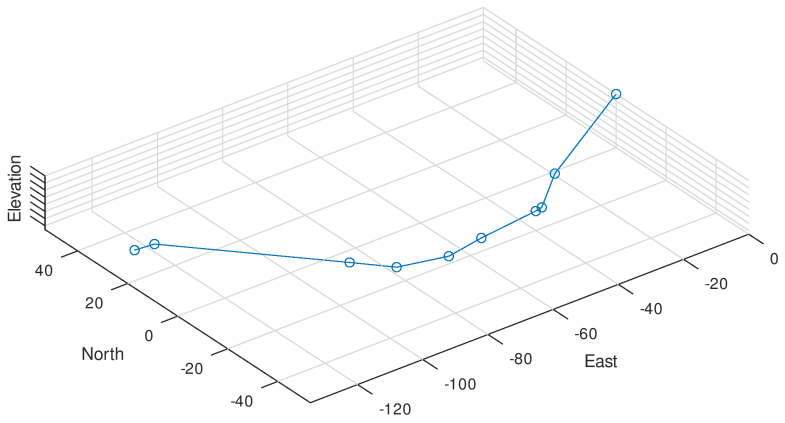


Figure 7.7: 3-dimensional model of survey line

Part 2: Passage modeling with cubic splines

We need additional data to model the passage outline. During the survey, estimates of the distances to the left wall, right wall, ceiling (up), and floor (down) were collected at each point. Cave surveyors call this "LRUD" data (i.e., left, right, up, down). There are ten survey stations and nine passage segments. Here is a table containing the LRUD data. The stations have been indexed 1 through 10, with 1 representing the entrance.

Station index	Left wall	Right wall	Ceiling (up)	Floor (down)
1	10.0	8.0	5.0	0.0
2	3.0	5.0	3.0	2.0
3	3.0	0.0	1.5	0.5
4	0.0	1.0	0.5	1.5
5	3.0	0.5	4.0	2.0
6	4.0	0.0	5.0	2.0
7	0.0	3.0	5.0	3.0
8	6.0	2.0	4.0	2.0
9	3.0	2.0	0.0	3.0
10	1.0	0.5	3.0	0.5

Since the cave segment consists of a single passage, oriented in an east-west direction, we can simplify things by partitioning the x-axis at the coordinates determined in problem 1. We use a model based on cubic splines[4].

The following script will generate cubic "Hermite" spline models for the walls, floor, and ceiling. The script and all required data files can be downloaded from https://gist.github.com/jalachniet/.

Octave Script 7.2: Cubic spline model

```
% script file 'cave_spline.m'
%   generates spline curves for walls, ceiling, and floor
%   plots plan and profile views of survey line and spline model
%   requires 10x1 station coordinate vectors x, y, and z (from part 1)
%   and 10x4 lrud data matrix (available in the file csurvey_data.txt)

% extract left, right, up, down from lrud data matrix
load csurvey_data.txt
L = lrud(:, 1);
R = lrud(:, 2);
U = lrud(:, 3);
D = lrud(:, 4);

% calculate left and right walls, floor and ceiling
y_left = y - L;
y_right = y + R;
z_down = z - D;
z_up = z + U;
```

[4] see http://mathworld.wolfram.com/CubicSpline.html and https://en.wikipedia.org/wiki/Cubic_Hermite_spline

```
19
20 % generate cubic spline models as piecewise polynomials
21 pp1 = interp1(x, y_left, 'pchip', 'pp');
22 pp2 = interp1(x, y_right, 'pchip', 'pp');
23 pp3 = interp1(x, z_down, 'pchip', 'pp');
24 pp4 = interp1(x, z_up, 'pchip', 'pp');
25
26 % extract and display breaks and coefficients
27 [breaks1, coeffs1] = unmkpp(pp1);
28 [breaks2, coeffs2] = unmkpp(pp2);
29 [breaks3, coeffs3] = unmkpp(pp3);
30 [breaks4, coeffs4] = unmkpp(pp4);
31 disp('Breaks ='), disp(breaks1(1 : size(breaks1, 2) - 1)')
32 disp('Left wall coefficients = '), disp(coeffs1)
33 disp('Right wall coefficients = '), disp(coeffs2)
34 disp('Floor coefficients = '), disp(coeffs3)
35 disp('Ceiling coefficients = '), disp(coeffs4)
36
37 % plot plan view
38 figure(1);
39 xx = linspace(min(x), max(x), 50);
40 yy1 = ppval(pp1, xx);
41 yy2 = ppval(pp2, xx);
42 plot(x, y, 'ro-', xx, yy1, 'linewidth', 2, 'b', xx, yy2, 'linewidth', 2, 'b
       ');
43 grid on;
44 axis('equal');
45 title('Skunk Cave - Plan View')
46 legend('survey line', 'cubic spline model')
47 xlabel('East (feet)')
48 ylabel('North (feet)')
49
50 % plot profile view
51 figure(2);
52 zz1 = ppval(pp3, xx);
53 zz2 = ppval(pp4, xx);
54 plot(x, z, 'ro-', xx, zz1, 'linewidth', 2, 'b', xx, zz2, 'linewidth', 2, 'b
       ');
55 grid on;
56 axis('equal');
57 title('Skunk Cave - Profile View')
58 legend('survey line', 'cubic spline model')
59 xlabel('East (feet)')
60 ylabel('Elevation (feet)')
```

This code will produce piecewise polynomials for the walls, floor, and ceiling as shown in Figures 7.8 and 7.9. These consist of 36 individual cubic polynomials for the walls, floor and ceiling, each with the form:

$$y = a(x - x_0)^3 + b(x - x_0)^2 + c(x - x_0) + d$$

where the x_0-values correspond to the x-coordinate breaks.

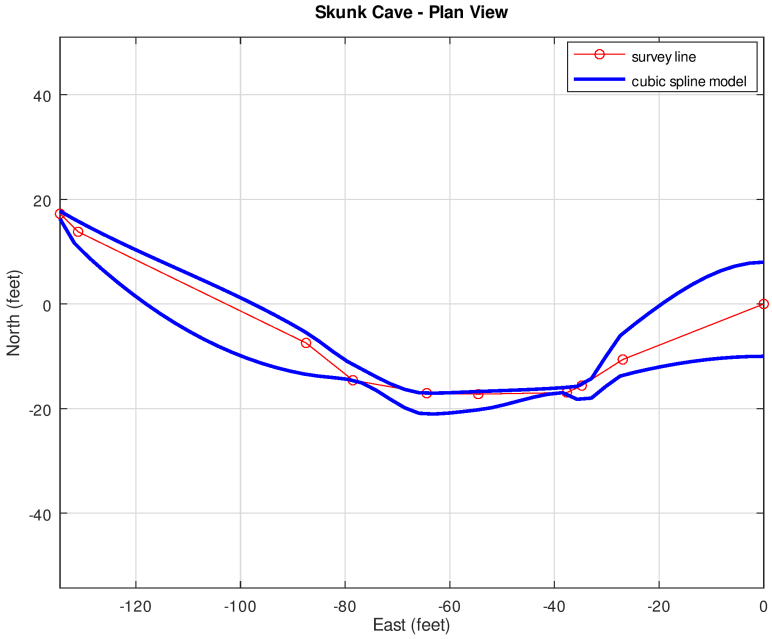


Figure 7.8: Cubic spline plan view

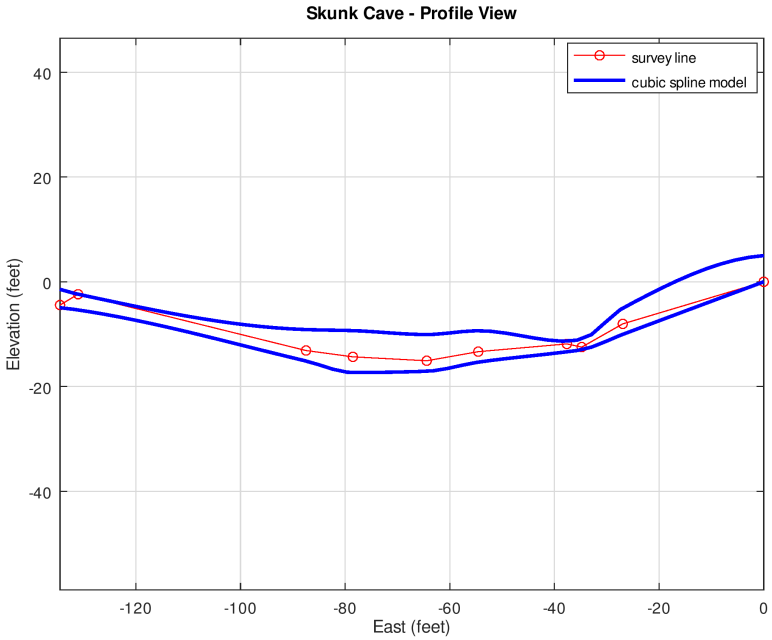


Figure 7.9: Cubic spline profile view

5. For each passage segment, the floor area can be calculated as

$$\int_a^b [\text{right wall} - \text{left wall}]\ dx$$

Use integration to calculate the floor area of each passage segment, and the total floor area. You may want to define the integrand using the ppval function, then integrate using quad.

6. The average height can be found using

$$h = \frac{1}{b-a}\int_a^b [\text{ceiling} - \text{floor}]\ dx$$

Calculate the average passage height in each segment.

7. A volume estimate for each segment can be calculated as floor area times average passage height, $V = Ah$. Use the floor area and passage height estimates to estimate the total volume of the entire passage.

8. A double integral can be used to directly calculate the volume between the floor and ceiling functions over the irregular plane region defined by the walls of the passage. Write a script to evaluate:

$$\sum_{i=1}^{9} \int_a^b \int_{\text{left wall}}^{\text{right wall}} [\text{ceiling} - \text{floor}]\ dA$$

Part 3: 3D-solid model

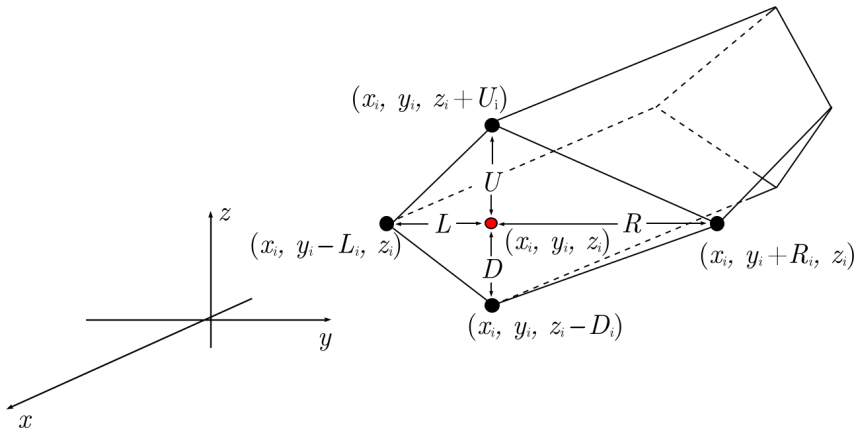


Figure 7.10: Kite-shaped cross section

Our goal now is to produce a true three-dimensional model, suitable for 3D-printing. Modeling a cave in three dimensions is a serious challenge because the passage not only winds around through space, its size and shape is also changing as you move along the passage. The most rudimentary approach is a "kite" shaped cross section, as shown above, obtained by fixing points in space to the left and right and above and below each station on the survey line.

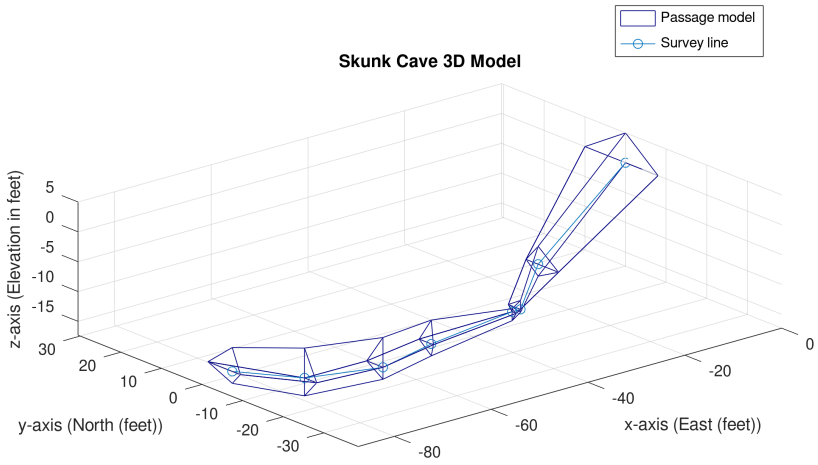


Figure 7.11: Skunk Cave - solid model

To graph the 3D model in Octave, we use an irregular meshgrid defined by the points outlined in Figure 7.10. Load the LRUD data file and extract the columns for L, R, U, and D. Then the mesh we need is built as:

```
>> X = [x x x];
>> Y = [y - L, y, y + R];
>> Z1 = [z, z + U, z];
>> Z2 = [z, z - D, z];
```

Here Z_1 is the ceiling and Z_2 is the floor. Now we merge these into a single set of meshgrid variables.

```
>> XX = [X X];
>> YY = [Y Y];
>> ZZ = [Z1 Z2];
```

Now surf or mesh can be used to plot the model using the XX, YY, and ZZ arrays. To show the survey line and passage wall model together, plot the walls with mesh and the following options.

```
>> mesh(XX, YY, ZZ);
>> hidden off;
>> axis('equal')
>> hold on;
>> plot3(x, y, z, 'o-');
```

The result should look something like Figure 7.11.

9. Follow the steps outlined above to create a 3D-model of the passage.

10. Use the methods shown in Section 7.4 to convert the resulting meshgrid variables into a printable STL file.

11. Pick a single passage segment and use double integration to calculate its volume. Start by producing a 3D model that shows just the single section of the cave. You can do this by repeating the steps shown above for the complete model, but selecting only the data for the relevant segment, from i to j:

```
>> X = [x(i:j) x(i:j) x(i:j)];
>> Y = [y(i:j) - L(i:j), y(i:j), y(i:j) + R(i:j)];
>> Z1 = [z(i:j), z(i:j) + U(i:j), z(i:j)];
>> Z2 = [z(i:j), z(i:j) - D(i:j), z(i:j)];
>> XX = [X X];
>> YY = [Y Y];
>> ZZ = [Z1 Z2];
```

    Then plot your section using the mesh command. For example, the section from 1 to 2 is shown in Figure 7.12. At first glance, you might think that the floor and ceiling sections could be modeled by planes. However, further investigation reveals that the boundaries of the "faces" are actually skew lines and therefore not contained in a plane.

    Next, look closely at the plan view. This can be plotted as x vs. y, or you can simply rotate the 3D block to the proper orientation. This view will be used to determine the limits of integration. As an example, the relevant domain for the segment from station 1 to 2 is shown in Figure 7.13.

    As noted, the two regions of the domain are not bounded above and below by planes (the four corners are not coplanar). To overcome this problem, you must further split the domain into four triangular subregions. Label the coordinates of each vertex and determine the equations in the xy-plane for the five sloping lines that divide the region into the union of four Type I regions. These should be reduced to $y = f(x)$ form.

    The volumes we need to compute are bounded between planar triangular floor and ceiling elements. You must determine the equations of planes representing the floor and the ceiling over each of the four regions. To write the equations, you will need to construct a pair of vectors in each plane, then use their cross product to write the normal vector. After simplifying, the equations should be written in the form $z = f(x, y)$. We have now effectively "triangulated" our irregular solid.

    The subvolume for each of the four regions can now be calculated using a double integral as follows:

$$\iint_R \Big[f_{\text{ceiling}}(x,y) - f_{\text{floor}}(x,y) \Big] \, dy \, dx$$

    The inner limits are determined by the lines that divide the domain and the outer limits are determined by the partition of the x-axis at each survey station.

    Calculate the four integrals to determine the volume enclosed.

12. Write a script that repeats the above steps for each passage segment, totaling the subvolumes to obtain an estimate for the full volume of the passage. (This is not a simple problem!)

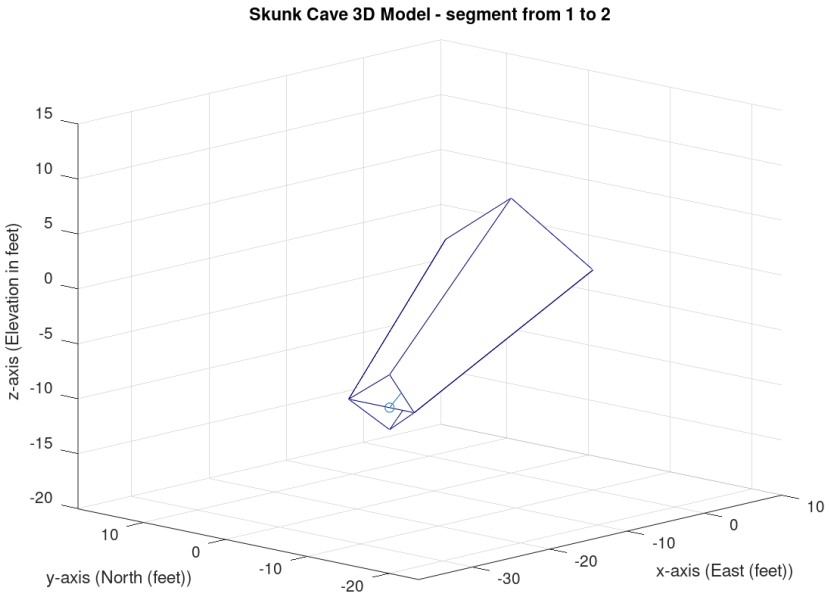


Figure 7.12: Skunk Cave - station 1 to 2 solid model

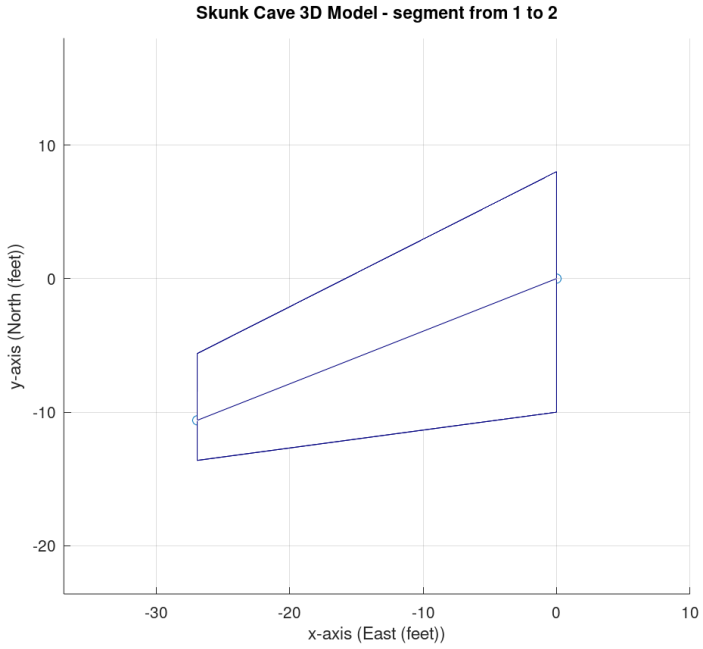


Figure 7.13: Skunk Cave - station 1 to 2 plan view

7.6 Modeling the spread of an infectious disease

Given the devastating impacts of the recent coronavirus pandemic, the mathematical models used to study the spread of infectious disease have lately been receiving considerable attention. The tools we have developed to this point are adequate for us to do some interesting simulations, as well as solve the standard differential equations model that is the basis of the widely circulated graphs depicting the outcome of "flattening the curve" through social distancing measures.

Unconstrained exponential growth

The simplest model is based on the so-called *basic reproduction number*. This is the number of new infections caused by each infected individual, commonly designated R_0. Its value is of critical importance in the study of the spread of infectious diseases. Suppose a disease is spreading through a population. One individual is infected initially and causes another R_0 infections, each of whom goes on to cause another R_0 infections, and so on. Supposing it takes t units of time for an individual to cause another R_0 infections, and with no further constraints, the number of infections will grow according to the model $f(t) = (R_0)^t$. Thus, for any disease with $R_0 > 1$, growth will be exponential.

1. Suppose a new disease enters a large population. Begin with one infection and assume that once infected, it takes one week for an individual to cause another two infections. How many infections are expected after six weeks? If growth continues unchecked, how long will it take until the number of cases exceeds 1000?

2. Suppose news reports indicate that in a certain locality the number of cases of a new disease is doubling every five days. Again assuming it takes one week for an individual to cause another R_0 infections, estimate the value of R_0. If there are currently 500 cases, project how many cases are expected in one month.

Simulation: constrained logistic growth

Nothing in nature can continue to grow exponentially without bound. If the population is finite, there is a limited carrying capacity and the exponential growth will lag, asymptotically approaching the carrying capacity. The characteristic curve is described by the logistic growth equation we considered in Chapter 3 Exercise 9.

Here is the scenario we will model: Suppose that a (benign) disease has entered a population of size N. The disease spreads according to following simplified assumptions:

- To begin, one person is chosen at random and infected.
- Once infected, an individual remains infectious indefinitely.
- In each subsequent time period, each infected individual comes into contact with one other person, chosen at random, and infects them (if they are not already infected).

The maximum number of infections is constrained by the population size N. In a classroom setting, it is instructive to run this simulation live among the population of students, tracking as the number of infections increases until the entire class is afflicted. But for now, we will use Octave to run a computer simulation, choosing a population size $N = 100$.

3. Start by creating a 1×100 vector of zeros, then randomly assign a value of 1 to a single position in the vector. Here we are using 0 to mean susceptible (not yet infected) and 1 to mean infected.

```
>> x = zeros(1, 100);              % vector of zeros
>> x(floor(rand*100) + 1) = 1; % set a random entry equal to 1
```

We can count the number of infected individuals by counting the number of 1s in our population vector. A convenient way to do this is with the equality logical test and a sum function.

```
>> I = sum(x==1) % count total number of entries equal to 1
I =  1
```

Now, we need to develop a loop such that each infected person comes into contact with someone at random, infecting them (if they are susceptible). We can use a while loop that cycles until the number infected reaches the upper limit of 100, along with a for loop to generate each new random case. To facilitate editing the code and making modifications to the parameters, it is advisable to put your code in an .m-file.

```
>> disp(I)
>> while I<100
     for k = 1:I
       x(floor(rand*100) + 1) = 1;
       I = sum(x==1);
     end
     disp(I)
   end
```

The disp statement will display the current number of infections at the end of each iteration. Enter the code and view the results.

4. We can improve the algorithm by adding a counter and a variable length vector that tracks the number of infections through each iteration. First reset the population vector **x**. Then try the following:

```
>> I = sum(x==1);
>> j = 1; a(j) = I;
>> while I<100
     j = j + 1;
     for k = 1:I
       x(floor(rand*100) + 1) = 1;
       I = sum(x==1);
     end
     a(j) = I;
   end
```

Run the simulation. The number of infections in each time period is recorded in the vector **a**. Create a plot showing the number of infections in each time period. Does the pattern appear to be logistic?

5. Use the nonlinear curve-fitting method explained in Section 7.2 Exercise 5 to fit a logistic model of the form

$$f(t) = \frac{C}{1 + Ae^{-kt}}$$

for the number of infections.

Here is the basic syntax to use, assuming vector **a** contains the results of your simulation. Its length is variable, depending on the random simulation, so in the code below we use the numel function for the upper limit on our input vector.

```
>> % load the optim package
>> pkg load optim

>> % define the model equation; p has three components (C, A, k)
>> f = @(p, t) p(1)./(1 + p(2).*exp(-p(3)*t));

>> % initial guesses for C, A, and k
>> init = [100; 200; 1];

>> % calculate the model
>> [p, y] = nonlin_curvefit(f, init, [1:numel(a)], a)
```

Plot the logistic curve on the same axes as the simulation data. Include a legend. Is the model a good fit?

The SIR model

With most diseases, an infected individual will not remain infectious indefinitely. They will either recover or succumb to the illness. In either case, they are removed from the population of susceptible individuals, at least assuming recovery provides a measure of immunity. Thus we now track three groups in the population: those susceptible, those infected, and those recovered (or removed). This is known as the *SIR model*.[5]

In a population of size N, let S be the number of individuals susceptible, I the number infected, and R the number recovered. If the infectious rate is β and the recovery rate is γ, then the SIR model corresponds to the following set of ordinary differential equations.

$$\frac{dS}{dt} = -\frac{\beta SI}{N} \tag{7.1}$$

$$\frac{dI}{dt} = \frac{\beta SI}{N} - \gamma I \tag{7.2}$$

$$\frac{dR}{dt} = \gamma I \tag{7.3}$$

In this model, β corresponds to the frequency of contact between individuals in the population and $\gamma = 1/d$, where d is the duration of the infection.

[5] https://en.wikipedia.org/wiki/Compartmental_models_in_epidemiology#The_SIR_model

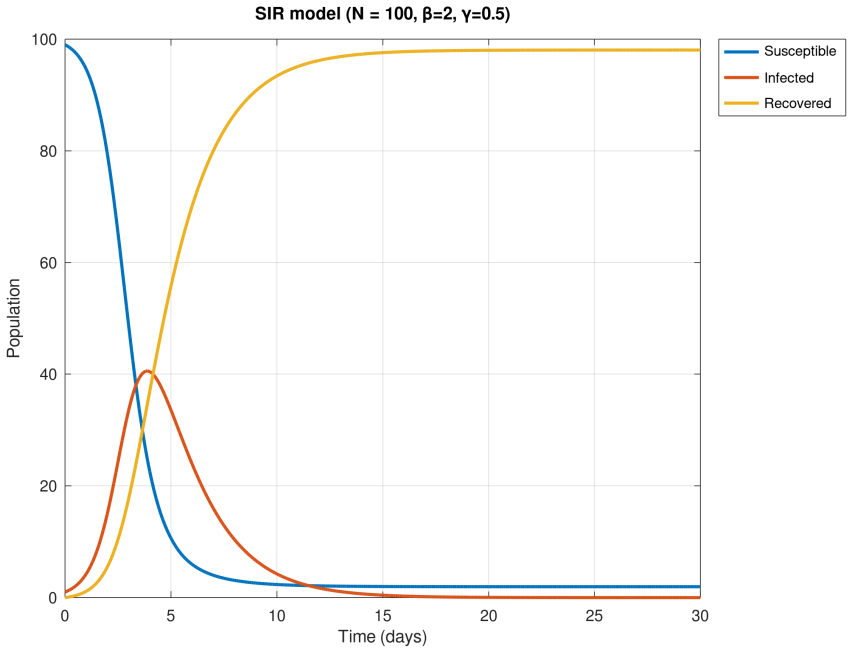


Figure 7.14: SIR model

6. Suppose we begin with one infection in a population of size $N = 100$ and the disease spreads according to the Equations 7.1–7.3 with $\beta = 2$ and $\gamma = 0.5$. Solve the system of differential equations over a 30-day period using ode45.

   We need to define y as a three-component vector function. Here is the syntax to use:

```
>> % enter the parameters
>> n = 100; b = 2; g = 0.5;

>> % define the ODE system
>> f = @(t, y) [-b*y(1).*y(2)/n; b*y(1).*y(2)/n - g*y(2); g*y(2)];
>> % y(1) = susceptible
>> % y(2) = infected
>> % y(3) = recovered

>> % solve the ODE system
>> tspan = [0 : 0.1 : 30]; % reduced step size for a smoother graph
>> init = [n - 1, 1, 0];   % begin with one infection
>> [t, y] = ode45(f, tspan, init);
```

   Plot a graph of the solution showing each of the three groups, labeled with a legend. You should get a result similar to Figure 7.14. Estimate the maximum number of infections and when it occurs.

7. Solve the system again, keeping N and γ as above, but changing β to 4. This corresponds to doubling the contact between individuals (that is, reduced social distancing). Plot a graph of the solution showing each of the three groups. Estimate the maximum number of infections and when it occurs.

8. What is effect of increased social distancing? Solve the model again, this time with $\beta = 1$ (keeping $N = 100$ and $\gamma = 0.5$). Plot a graph of the solution showing each of the three groups. Estimate the maximum number of infections and when it occurs.

9. Now plot a comparative graph showing only the number of infections, with $\beta = 1$, $\beta = 2$, and $\beta = 4$. Include a legend. The graph should clearly illustrate the effect reduced contact between individuals has on the timing and magnitude of the peak number of infections.

   In the context of this model, the basic reproduction number is $R_0 = \beta/\gamma$. Give the basic reproduction number for each of the three models under consideration.

10. Write a function file that solves the SIR model equations for any user-inputted values of N, β, and γ. You will also need to specify the interval over which the equations are solved.

    Your function should have the following basic structure:

Octave Script 7.3: SIR model function file

```
function y = SIRmodel(n, b, g, maxT)
  % SIR model for spread of infectious disease
  % y = SIRmodel(n, b, g, maxT)
  %   parameters:
  %     n = population size
  %     b = transmission rate (beta)
  %     g = recovery rate (gamma)
  %     maxT = number of iterations to simulate
  %
  %   example:
  %     SIRmodel(100, 3, 0.5, 20)

   ---body of function---

end
```

The output should be a matrix showing the values for S, I, and R in three columns. Also, generate a plot, labeled with a legend and title (like Figure 7.14). Run the function for various values of the parameters to explore their effect on the solution. Try to find scenarios where the disease runs its course without affecting the entire population.

Appendix A

MATLAB compatibility

Octave and MATLAB use similar syntax, but the programs are not identical. MATLAB, especially when extended with its various toolboxes, has many functions not available in Octave. However, most code in this book will work in MATLAB, so what you've learned here will easily transfer to more advanced MATLAB programming.

Octave allows some flexibility in syntax that MATLAB does not. This book has been written with MATLAB compatibility in mind, so generally, when multiple forms of a command or operation are possible, the MATLAB-compatible option has been used. Here is a summary of a few of the potential coding differences.

- Comments in Octave can be preceded by # or %. MATLAB uses %.
- Octave recognizes single quotes and double quotes around strings. MATLAB requires single quotes.
- Blocks in Octave can be terminated with statements based on the initial command (endfor, endfunction, etc.). MATLAB uses only end.
- The not-equal comparison can be written as != or ~= in Octave. MATLAB requires ~=.
- Octave supports C-style increment operators. For example, in Octave, ++n is equivalent to n = n+1, but in MATLAB, increment operators like ++ are not available.
- Octave allows user-defined functions to be entered at the command line or in scripts. MATLAB requires the use of separate function files (recent releases now allow defining functions in scripts).
- MATLAB users may prefer fplot and fimplicit over the ezplot function used in Sections 3.3.2 and 3.4.3. However, fimplicit is not (yet) implemented in Octave and the current implementation of fplot does not handle symbolic functions. Octave's fplot does work for plotting functions of the form $y = f(x)$.
- MATLAB does not allow the formatted output command printf. Use fprintf.

- For systems that are over or underdetermined, and those with a singular coefficient matrix, left division in Octave will always return a minimum norm solution, equivalent to solving using the pseudoinverse. The result may not be consistent with MATLAB's solution (refer to MATLAB documentation for details on how MATLAB-left division handles these cases).
- The lsode command is not implemented in MATLAB. For MATLAB-compatibility, ode45 and several other MATLAB-compatible solvers are available in Octave (some require loading the package odepkg).
- There are significant differences between MATLAB Toolboxes and Octave Forge packages, though many Octave packages use syntax similar or identical to the corresponding MATLAB functions. In particular, the packages used used in this text have good correlation to MATLAB Toolboxes: the Octave symbolic package is similar to MATLAB's Symbolic Toolbox, the optim package has capabilities similar to the Curve Fitting Toolbox, the image package is similar to the Image Processing Toolbox, and the statistics package is similar to the Statistics Toolbox.

Appendix B

Octave command glossary

The names and basic syntax for many common commands are provided below. Many commands have additional options. Type help NAME at the Octave prompt or refer to [3] for more details. Note that some commands listed here require Octave Forge packages, including statistics and symbolic.

List of Octave commands

Syntax	Description
[a:b]	vector from a to b by increment 1 unit
[a : step : b]	vector from a to b by increment "step"
A'	(conjugate) transpose of A
A.'	transpose of A
A*B	matrix product AB
A\b	left division, solves system $A\mathbf{x} = \mathbf{b}$
A^n	matrix power
A + B	sum $A + B$
x < y	less than comparison
x > y	greater than comparison
x == y	equality comparison
x ~= y	not equal comparison
x.*y	elementwise product
x./y	elementwise quotient
x.^n	elementwise exponent
a & b	logical AND
~a	logical NOT
a | b	logical OR
f = @(x1, x2, ...) rule	anonymous function of $x_1, x_2, \ldots$ given by rule
abs(x)	absolute value or modulus of x
acos(x)	inverse cosine of x in radians
angle(z)	angle of complex variable z
ans	result of last calculation
asin(x)	inverse sine of x in radians
assume(x, 'property')	assume symbolic x has property (positive, integer, etc.)
atan(x)	inverse tangent of x in radians
axis([Xmin Xmax Ymin Ymax])	set axis limits
bar(x, y)	bar graph of y vs. x
besselj(n, x)	order n Bessel functions of the first kind
binopdf(x, n, p)	binomial probability of x successes
ceil(x)	$\lceil x \rceil$, the least integer greater than or equal to x

continued ...

. . . continued

Syntax	Description
clc	clear command window
clear var1 var2 ...	clear variables (clear all if no variable listed)
clf	clear plot window
colormap('type')	set colormap to "type" (e.g., gray, default)
comet(x, y)	comet plot animation of a plane curve
comet3(x, y, z)	comet plot animation of a space curve
compass(z)	compass plot of variable z in the complex plane
contour(X, Y, Z)	contour plot of surface
corr(x, y)	linear correlation coefficient r
cos(x)	cosine of x (x in radians)
cosh(x)	hyperbolic cosine of x
cross(u, v)	cross product of vectors **u** and **v**
csvread('filename.csv')	load a CSV-format data file
csvwrite('filename.csv', A)	write a numeric matrix A to a CSV-format data file
curl(X, Y, U, V)	curl of vector field with components U and V
cylinder([r r])	cylinder of radius r
dblquad('f', a, b, c, d)	double integral over rectangle
det(A)	determinant of A
diff(f, x)	differentiate symbolic function with respect to x
dir	list files in current directory
disp(x)	display the value of x
divergence(X, Y, U, V)	divergence of vector field with components U and V
dot(u, v)	dot product of vectors **u** and **v**
double(x)	convert x to double precision floating point
dsolve(ode, ic)	solve symbolic differential equation with initial condition
e	the number e
[v lambda] = eig(A)	find eigenvalues and eigenvectors
erf(x)	the error function
exp(x)	natural exponential function
expand(f)	expand a symbolic expression
eye(n)	$n \times n$ identity matrix
ezplot(f, [a b c d])	implicit plot of $f(x, y) = 0$ over domain $[a, b] \times [c, d]$
factor(f)	factor a symbolic expression
factorial(n)	$n!$, factorial of n
floor(x)	$\lfloor x \rfloor$, the greatest integer less than or equal to x
for k = 1:n ... end	for loop
format opt	decimal format, options = short, long, free, bank, etc.
fplot(f, [a b])	plot of $y = f(x)$ over domain $[a, b]$
fsolve('f', x1)	solve $f(x) = 0$, initial guess x_1
function y = f(x) ... end	define a function
gamma(x)	the gamma function
[DX DY] = gradient(Z)	gradient of a vector field
grid on/off	toggle plot grid

continued . . .

. . . continued

Syntax	Description
help NAME	get documentation for command "NAME"
hist(X, B)	histogram of X, optional B specifies bins
hold on/off	add to current plot toggle on/off
i/I/j/J	the imaginary unit
imag(z)	imaginary part of z
imagesc(A)	display matrix as scaled image
int(f, x, a, b])	integrate symbolic function (optional limits)
interp1(x, y, 'method')	interpolate using type "method" (linear, cubic, pchip, etc.)
inv(A)	inverse of matrix A
legend('plot1', 'plot2', ...)	plot legend
linspace(a, b, n)	vector of n evenly spaced points from a to b
load filename	load saved variables
log(x)	natural logarithm
lsode('f', x0, t)	solves $dx/dt = f(x,t)$, $x(0) = x_0$
[L U P] = lu(A)	LU decomposition of A, with permutation
max(A)	maximum of vector, or column-wise maximums of a matrix
mean(x)	mean of x
mesh(X, Y, Z)	surface plotted as a mesh
[X Y] = meshgrid(x, y)	generate xy-meshgrid
min(A)	minimum of vector, or column-wise minimums of a matrix
norm(u)	norm (length) of vector $\mathbf{u}$
normcdf(x, mu, sigma)	area under normal curve to the left of x
norminv(a, mu, sigma)	inverse normal distribution given area a
nthroot(number, index)	real nth root
numel(v)	number of elements in a vector or matrix
[t x] = ode45('f', tspan, x0)	solve $dx/dt = f(t,x)$ with initial condition $x(0) = x0$
ones(m, n)	$m \times n$ matrix of ones
peaks	example 3d graph of a surface with many local extrema
pi	the number π
pinv(A)	pseudoinverse of A
pkg install −forge NAME	download and install a package from Octave Forge
pkg list	list installed packages
pkg load NAME	load an installed package
plot(x, y)	plot of x vs. y
plot3(x, y, z)	plot space curve
polar(theta, rho)	polar plot of radial distance ρ vs. angle θ (in radians)
polyfit(x, y, order)	polynomial fit x vs. y of degree "order"
polyval(P, x)	evaluate polynomial P at x
ppval(P, x)	evaluate piecewise polynomial P at x
print −dpng filename.png	save plot as png (substitute jpg, eps, etc.)
pwd	print (list) current working directory
[Q R] = qr(A)	QR decomposition of A
quad('f', a, b)	definite integral of f from a to b

continued . . .

...continued

Syntax	Description
quiver(X, Y, U, V)	plot vector field with components U and V
quiver3(X, Y, Z, U, V, W)	plot 3d vector field with components U, V, W
rand(m, n)	$m \times n$ random matrix (uniformly distributed entries)
randn(m, n)	$m \times n$ random matrix (normally distributed entries)
rank(A)	rank of A
real(z)	real part of z
save filename A B	save variables A, B, ...
set(h, 'property', 'value')	set properties of graphics object h
simplify(f)	simplify a symbolic expression
sin(x)	sine of x (x in radians)
sinh(x)	hyperbolic sine of x
size(A, opt)	dimensions of A; dimension option: 1=rows, 2=cols
solve(a == b)	solve a symbolic equation
sqrt(x)	principal square root of x
std(x)	standard deviation of x
subs(f, x)	evaluate symbolic expression f at x
sum(A)	sum of vector components or column-wise sum of matrix A
surf(X, Y, Z)	surface plot
[U S V] = svd(A)	singular value decomposition of A
syms x y z	define symbolic variables
tan(x)	tangent of x (x in radians)
tanh(x)	hyperbolic tangent of x
taylor(f, x, a, 'order', n)	degree n Taylor polynomial of f about a
tcdf(x, n)	Students t-distribution CDF with degrees of freedom n
text(x, y, 'label')	add a text label to plot at coordinates (x, y)
tinv(a, n)	inverse t-distribution with degrees of freedom n
title('name')	assign plot title
triplequad('f', a, b, c, d, e, f)	triple integral over a rectangular box
ttest(X, mu, 'name', 'value') ...	t-test, name-value pairs set alpha and tail (left, right, both)
[x, p] = unmkpp(P)	extract components of piecewise polynomial P
var(x)	variance of x
while (condition) ... end	while loop
whos	list variables in current scope
xlabel('name')	horizontal axis label
ylabel('name')	vertical axis label
zeros(m, n)	$m \times n$ matrix of zeros

Note that most commands which accept a function as an argument expect the name to be in quotes ('f') if it is defined as a named function (using the function ... end construction, whether so-defined at the command line, in a script, or in a function file), but if f is a function *handle* (e.g., defined as anonymous function), then no quotes are used.

References

[1] Brin, Leon Q, *Tea Time Numerical Analysis, 2nd edition.* CC-BY-SA, 2016.
http://lqbrin.github.io/tea-time-numerical/

[2] Diez, David M, Christopher D Barr, and Mine Çetinkaya-Rundel, *OpenIntro Statistics, 3rd edition.* CC-BY-SA, 2015.
https://www.openintro.org/stat/textbook.php

[3] Eaton, John W, David Bateman, Søren Hauberg, and Rik Wehbring, *GNU Octave Manual: Edition 5.* GNU FDL, 2019.
https://www.gnu.org/software/octave/octave.pdf

[4] Grinstead, Charles M, and J Laurie Snell, *Introduction to Probability, 2nd Edition.* American Mathematical Society. GNU FDL, 2006.
http://www.dartmouth.edu/~chance/teaching_aids/books_articles/probability_book/book.html

[5] Hartman, Gregory, et al, *Apex Calculus, Version 4.0.* CC-BY-NC, 2018.
http://www.apexcalculus.com/

[6] Nicholson, W Keith, *Linear Algebra with Applications, Open Edition (Revision 2019A).* Lyryx. CC-BY-NC-SA, 2019.
https://lyryx.com/products/mathematics/linear-algebra-applications/

[7] Strang, Gilbert, Edwin Herman, et al, *Calculus Volume 3.* OpenStax. CC-BY-NC-SA, 2016.
https://openstax.org/details/books/calculus-volume-3

[8] Trench, William F, *Elementary Differential Equations.* Trinity University, Digital Commons. CC-BY-NC-SA, 2013.
http://digitalcommons.trinity.edu/mono/8/

Index

antiderivative, 53
arc length, 58, 133
assumption on variable, 54
axis
    equal aspect ratio, 46
    grid, 10
    implicit plot limits, 49
    labels, 10
    limits, 10, 100
    off, 15

bar graph, 68
Bessel function, 61
binomial distribution, 68
Boolean operators, 108
bumpy sphere, 104, 141

central limit theorem, 63
change of variables, 102
clear
    plot window, 11
    variables, 14
code listing, format, x
color map, 99
command history, 3
comments, x, 4
compass plot, 60
complex number, 8, 59
computer algebra system, 2, 50
constant function, 15
contour plot, 100
correlation coefficient, 66
cross product, 5, 137
curl, 125
curvature, 134
curve-fitting
    exponential, 36
    general nonlinear, 131, 154
    linear, 11, 23, 66, 86
    polynomial, 25
cycloid, 46
cylinder, 124, 134
cylindrical coordinates, 102

dblquad, 108
derivative, 53
determinant, 8
diagonalization, 78
difference quotient, 51
differential equation, 117, 120, 154, 158
dilation, 31
disp statement, 153
display format, 40, 65
divergence, 125
dot product, 5, 89, 91

eigenvalues/eigenvectors, 8, 73, 81
elementwise operations, 12, 24, 39, 97
equality comparison, 51
equilibrium vector, 76
error function, 61
Euler's method, 118
Eulerian path, 27
evaluating symbolic expressions, 51
exponential function, base e, 15, 43

factorial, 61
factoring, 51
file browser, 3
floating point format
    free, 65
    long, 18
floor function, 34, 63
for loop, 41, 44, 153
formatted output, 67, 157
fsolve, 57
function
    anonymous, 39, 110
    file, x, 66, 88, 111, 137
    symbolic, 52
    user-defined, 43

gamma function, 61, 70
Gaussian elimination, 17
Gini index, 131
GNU, 1
gradient field, 115
Gram-Schmidt process, 88
graphing, *see* plotting

harmonic series, 42
helix, 97, 133
histogram, 63
hold on/off, 11
homogeneous coordinates, 32, 37
hypothesis test, 69

identity matrix, 8
ill-conditioned system, 85
image processing, 128
imaginary number, 59
implicit function, 47
import data, *see* load
integral
    multiple, 108
    numeric, 43
    symbolic, 53
interpolation, 145

least-squares solution, 24, 34, 85
left division, 19, 33, 158
length of vector, 5
limaçon, 46
limit, 39
linear regression, *see* curve-fitting,linear
linear system, *see* system of linear equations
linear transformation, 32
linspace, 9
load
    comma-separated data, 16
    Octave workspace, 3
    saved variable(s), 3
logical function, 108, 153
logistic growth, 58, 126, 152
Lorenz curve, 130
lsode, 120
LU decomposition, 19, 33

m file, *see* Octave script
Markov chain, 74
MATLAB, ix, 1, 157
matrix
    definition, 5
    dilation, 31
    identity, 8
    indexing, 17
    inverse, 8
    lower triangular, 20
    multiplication, 7
    of ones, 15, 24, 115
    of zeros, 3
    orthogonal, 80
    permutation, 22
    random, 34
    reflection, 29
    rotation, 28
    row echelon form, 18, 20
    row-reduced echelon form, 18
    singular, 33, 158
    symmetric, 80
    transpose, 8
    upper triangular, 20
mean, 63
mesh, 100
mesh2stl, 137
meshgrid, 99
midpoint rule, 43, 125

natural logarithm, 36
nested loops, 113
Newton's method, 57
nonrectangular domain, 102, 108
norm, 5
normal distribution, standard, 63
normal equations, 24
normalize a vector, 6
nth root (real), 60
numerical integration, 43, 108

Octave
    graphical user interface, 2
    installation, 2
    octaverc file, 3
    script, x, 3, 43, 44
ODE, *see* differential equation
ode45, 122
order of operations, 4
orthogonal diagonalization, 80
orthonormal set, 88

packages
    image, 128
    installing, 127
    loading, 127
    optimization, 131
    ordinary differential equations, 158
    statistics, 68, 69, 159
    symbolic, 50, 125, 141, 159
parametric surface, 105
partial sums, sequence of, 41
pchip, 145
peaks, 115, 140
permutation matrix, 22
plotting, 9
    comet animation, 46
    coordinate axes, 15
    ezplot, 47, 54, 157
    fplot, 157
    implicit, 47
    legend, 10
    plot options, 10, 13
    plot title, 10
    plotting points, 11
    polar, 47
    set function, 55
    surface, 99
    symbolic, 54
    text labels, 15
    three-dimensional, 97
    vector field, 114
polar coordinates, 46, 102
polynomial
    format, 26
    interpolation, 23, 25
    piecewise, 146
    polyfit function, 25, 66
    polyval function, 26, 66
printf, 67
printing to file, 13
probability vector, 75
projection
    scalar, 6
    vector, 6, 88
pseudoinverse, 85, 158
Python, 1, 50, 56

QR algorithm, 91
QR decomposition, 90
quadrature (definite integral), 43
quiver, 114, 117
quiver3, 114

rand function, 34
random integer, 34, 63
rank, 8
reflection, 29
regression, *see* curve-fitting
Riemann sum, 112
rotation, 28
rref command, 18

save variable(s), 3
scientific notation, 42, 60, 65
script, *see* Octave script
semi-log plot, 36
sequence, 41
set function, 55
Simpson's rule, 43
singular value decomposition, 82, 127
singular values, 82, 127
SIR model, 154
slope field, 117
solid of revolution, 107
solve symbolic equation, 51
sombrero function, 139
space curve, 97
special functions, 61
spherical coordinates, 104
spline curve, 145
startup.m, 3
statistics package, 68, 69
STL (STereoLithography), 136
string variables, x
suppress output, 9, 34, 42
surf command, 99
surface, 99
    triangulation, 137, 150
svd command, 85
symbolic package, 50
    help function, 56
SymPy, 50
system of linear equations, 17, 19, 21, 23, 86
    backward substitution, 18
    forward substitution, 21
    ill-conditioned, 85
    inconsistent, 33

infinitely many solutions, 34
left division, 19
LU decomposition, 19
over/underdetermined, 158

t-test, 69
Taylor series, 58
text function, 15
three-dimensional printing, 136
transition matrix, 75
transpose, 8
conjugate, 8
trapezoid rule, 43
triplequad, 108
tube plot, 142

uniform distribution, 34, 63

variable
assignment, 4
editor, 3
symbolic, 51
variance, 63
vector field, 114
vectorized code, 39, 45, 113

while loop, 153
workspace, 3

This text provides a brief, noncomprehensive introduction to GNU Octave, a free software alternative to MATLAB. The basic syntax and usage is explained through concrete examples from the mathematics courses a math, computer science, or engineering major encounters in the first two years of college: linear algebra, calculus, differential equations, and statistics.

Copyright 2020 by Jason Lachniet.

This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.

Download for free at:
https://www.wcc.vccs.edu/sites/default/files/Introduction-to-GNU-Octave.pdf